Math. Struct. in Comp. Science (2005), *vol.* 15(1), *pp.* 37–92.
DOI: 10.1017/S0960129504004426 Printed in the United Kingdom

Definitions by rewriting in the Calculus of Constructions

Frédéric Blanqui[1,2]

[1] *Laboratoire d'Informatique de l'École Polytechnique (LIX)*
91128 Palaiseau Cedex, France
(until 30 September 2003)
[2] *Institut National de Recherche en Informatique et Automatique (INRIA)*
Laboratoire lorrain de Recherche en Informatique et ses Applications (LORIA)
615 rue du Jardin Botanique, BP 101, 54602 Villers-lès-Nancy, France
blanqui@loria.fr *(from 1st October 2003)*

Received 16 September 2002. Revised 12 November 2003.

Abstract: *This paper presents general syntactic conditions ensuring the strong normalization and the logical consistency of the Calculus of Algebraic Constructions, an extension of the Calculus of Constructions with functions and predicates defined by higher-order rewrite rules. On the one hand, the Calculus of Constructions is a powerful type system in which one can formalize the propositions and natural deduction proofs of higher-order logic. On the other hand, rewriting is a simple and powerful computation paradigm. The combination of both allows, among other things, to develop formal proofs with a reduced size and more automation compared with more traditional proof assistants. The main novelty is to consider a general form of rewriting at the predicate-level which generalizes the strong elimination of the Calculus of Inductive Constructions.*

1. Introduction

This work aims at defining an expressive language allowing to specify and prove mathematical properties easily. The quest for such a language started with Girard' system F (Girard 1972) on the one hand and De Bruijn's Automath project (De Bruijn 1968) on the other hand. Later, Coquand and Huet combined both calculi into the Calculus of Constructions (CC) (Coquand 1985). As in system F, in CC, data types are defined through impredicative encodings that are difficult to use in practice. So, following Martin-Löf's theory of types (Martin-Löf 1984), Coquand and Paulin-Mohring defined an extension of CC with inductive types and their associated induction principles as first-class objects, the Calculus of Inductive Constructions (CIC) (Coquand and Paulin-Mohring 1988), which is the basis of the proof-assistant Coq (Coq Development Team 2002).

However, defining functions or predicates by induction is not always convenient. Moreover, with such definitions, equational reasoning is uneasy and leads to very large proof terms. Yet, for decidable theories, equational proofs need not to be kept in proof terms.

This idea that proving is not only reasoning (undecidable) but also computing (decidable) has been recently formalized in a general way by Dowek, Hardin and Kirchner with the Natural Deduction Modulo (NDM) for first-order logic (Dowek *et al.* 1998).

A more convenient and powerful way of defining functions and predicates is by using rewrite rules (Dershowitz and Jouannaud 1990). This notion is very old but its study really began in the 70's with Knuth and Bendix (Bendix and Knuth 1970) for knowing whether, in a given equational theory, an equation is valid or not. Then, rewriting was quickly used as a programming paradigm (see (Dershowitz and Jouannaud 1990)) since any computable function can be defined by rewrite rules.

In the following sub-sections, we present in more details our motivations for extending CIC with rewriting, the previous works on the combination of λ-calculus and rewriting, and our own contributions.

1.1. *Advantages of rewriting*

In CIC, functions and predicates can be defined by induction on inductively defined types. The case of the type nat of natural numbers, defined from $0 : nat$ (zero) and $s : nat \Rightarrow nat$ (successor function), yields Gödel' system T: a function $f : nat \Rightarrow \tau$ is defined by giving a pair of terms (u, v), written $(rec\ u\ v)$, where $u : \tau$ is the value of $f(0)$ and $v : nat \Rightarrow \tau \Rightarrow \tau$ is a function which computes the value of $f(n+1)$ from n and $f(n)$. Computations proceeds by applying the following (higher-order) rewrite rules, called *ι-reduction*:

$$\begin{array}{rcl} rec\ u\ v\ 0 & \to_\iota & u \\ rec\ u\ v\ (s\ n) & \to_\iota & v\ n\ (rec\ u\ v\ n) \end{array}$$

For instance, addition can be defined by the term $\lambda xy.(rec\ u\ v\ x)$ with $u = y$ and $v = \lambda nr.s(r)$ (definition by induction on x). Then, one can check that:[†]

$$\begin{array}{c} 2 + 2 \to^*_\beta rec\ 2\ v\ 2 \to_\iota v\ 1\ (rec\ 2\ v\ 1) \to^*_\beta s(rec\ 2\ v\ 1) \\ \to_\iota s(v\ 0\ (rec\ 2\ v\ 0)) \to^*_\beta s(s(rec\ 2\ v\ 0)) \to_\iota s(s(2)) = 4 \end{array}$$

Proofs by induction are formalized in the same way: if P is a predicate on natural numbers, u a proof of $P0$ and v a proof of $(n : nat)Pn \Rightarrow P(sn)$,[‡] then $rec\ P\ u\ v$ is a proof of $(n : nat)Pn$, and ι-reduction corresponds to the elimination of induction cuts. In fact, $(rec\ u\ v)$ is nothing but a particular case of $(rec\ P\ u\ v)$ with the non-dependent predicate $P = \lambda n.\tau$.

In addition, deduction steps are made modulo $\beta\iota$-equivalence[§], that is, if π is a proof of P and $P =_{\beta\iota} Q$, then π is also a proof of Q. For instance, if π is a proof of $P(2+2)$, then it is also a proof of $P(4)$, as one would naturally expect. The verification that a term π is indeed a proof of a proposition P, called type-checking, is decidable since $\beta\iota$ is a confluent (the order of computations does not matter) and strongly normalizing (there is no infinite computation) relation (Werner 1994).

[†] $\to^*_\beta$ is the transitive closure of the β-reduction relation: $(\lambda x.t\ u) \to_\beta u\{x \mapsto t\}$.

[‡] As often in type systems, we denote universal quantification over a type T by $(x : T)$.

[§] Reflexive, symmetric and transitive closure of the $\beta\iota$-reduction relation (which is the union of the β and ι reduction relations).

Although the introduction of inductive types and their induction principles as first-class objects is a big step towards a greater usability of proof assistants, we are going to see that the restriction of function definitions to definitions by induction, and the restriction of type conversion to $\beta\iota$-equivalence, have several important drawbacks. The use of rewriting, that is, the ability of defining functions by giving a set of rewrite rules $\mathcal{R}$, and the possibility of doing deductions modulo $\beta\mathcal{R}$-equivalence, can remedy these problems. It appears that ι-reduction itself is nothing but a particular case of higher-order rewriting (Klop *et al.* 1993; Nipkow 1991) where, as opposed to first-order rewriting, the constructions of the λ-calculus (application, abstraction and product) can be used in the right hand-sides of rules.¶ A common example of a higher-order definition is the function *map* which applies a function f to each element of a list:

$$\begin{array}{rcl} map\ f\ nil & \to & nil \\ map\ f\ (cons\ x\ \ell) & \to & cons\ (f\ x)\ (map\ f\ \ell) \end{array}$$

where *nil* stands for the empty list and *cons* for the function adding an element at the head of a list.

Easier definitions. First of all, with rewriting, definitions are easier. For instance, addition can be defined by simply giving the rules:

$$\begin{array}{rcl} 0 + y & \to & y \\ (s\ x) + y & \to & s\ (x + y) \end{array}$$

Then, we have $2 + 2 \to s(2 + 1) \to s(s(2 + 0)) \to s(s(2)) = 4$. Of course, one can make the definitions by induction look like this one, as it is the case in Coq (Coq Development Team 2002), but this is not always possible. For instance, the definition by induction of the comparison function $\leq$ on natural numbers requires the use of two recursors:

$$\lambda x.rec\ (\lambda y.true)\ (\lambda nry.rec\ false\ (\lambda n'r'.rn')\ y)\ x$$

while the definition by rewriting is simply:

$$\begin{array}{rcl} 0 \leq y & \to & true \\ s\ x \leq 0 & \to & false \\ s\ x \leq s\ y & \to & x \leq y \end{array}$$

More efficient computations. From a computational point of view, definitions by rewriting can be more efficient, although the process of selecting an applicable rule may have a higher cost (Augustsson 1985). For example, since $+$ is defined by induction on its first argument, the computation of $n + 0$ requires $n + 1$ reduction steps. By adding the rule $x + 0 \to x$, this takes only one step.

Quotient types. Rewriting allows us to formalize some quotient types in a simple way, without requiring any additional extension (Barthe and Geuvers 1995; Courtieu 2001), by simply considering rewrite rules on constructors, which is forbidden in CIC since constructors must be free in this system. For instance, integers can be formalized by taking 0 for zero, p for predecessor and s for successor, together with the rules:

¶ We will not consider higher-order pattern-matching here although it should be possible as we show it for the simply-typed λ-calculus in (Blanqui 2000).

$$\begin{array}{rcl} s\ (p\ x) & \to & x \\ p\ (s\ x) & \to & x \end{array}$$

This technique applies to any type whose constructors satisfy a set of equations that can be turned into a confluent and strongly normalizing rewrite system (Jouannaud and Kounalis 1986).

More automation. We previously saw that, in CIC, if P is a predicate on natural numbers, then $P(2+2)$ is $\beta\iota$-equivalent to $P(4)$ and, hence, that a proof of $P(2+2)$ is also a proof of $P(4)$. This means that proving $P(4)$ from $P(2+2)$ does not require any argument: this is automatically done by the system. But, because functions must be defined by induction, this does not work anymore for computations on open terms: since $+$ is defined by induction on its first argument, $P(x+2)$ is not $\beta\iota$-equivalent to $P(s(s(x)))$. Proving $P(s(s(x)))$ from $P(x+2)$ requires a user interaction for proving that $x+2$ is equal to $s(s(x))$, which requires induction.

We may even go further and turn some lemmas into simplification rules. Let us for instance consider the multiplication on natural numbers:

$$\begin{array}{rcl} 0 \times y & \to & 0 \\ (s\ x) \times y & \to & y + (x \times y) \end{array}$$

Then, the distributivity of the addition over the multiplication can be turned into the rewrite rule:

$$(x+y) \times z \quad \to \quad (x \times z) + (y \times z)$$

hence allowing the system to prove more equalities and more lemmas automatically by simply checking the $\beta\mathcal{R}$-equivalence with already proved statements. In the case of an equality $u = v$, it suffices to check whether it is $\beta\mathcal{R}$-equivalent to the instance $u = u$ of the identity axiom, which is the same as checking whether u and v have the same $\beta\mathcal{R}$-normal form.

Smaller proofs. Another important consequence of considering a richer equivalence relation on types is that it reduces the size of proofs, which is currently an important limitation in proof assistants like Coq. For instance, while the proof of $P(s(s(x)))$ requires the application of some substitution lemma in CIC, it is equal to the proof of $P(x+2)$ when rewriting is allowed. The benefit becomes very important with equality proofs, since they require the use of many lemmas in CIC (substitution, associativity, commutativity, etc.), while they reduce to reflexivity with rewriting (if one considers rewriting modulo associativity and commutativity (Peterson and Stickel 1981)).

More typable terms. The fact that some terms are not $\beta\iota$-equivalent as one would expect has another unfortunate consequence: some apparently well-formed propositions are rejected by the system. Take for instance the type $list : (n : nat)\star$ of lists of length n with the constructors $nil : list0$ and $cons : nat \Rightarrow (n : nat)listn \Rightarrow list(sn)$. Let $app : (n : nat)listn \Rightarrow (n' : nat)listn' \Rightarrow list(n + n')$ be the concatenation function on $list$. If, as usual, app is defined by induction on its first argument then, surprisingly, the following propositions are not typable in CIC:

$$\begin{aligned} app\ n\ \ell\ 0\ \ell' &= \ell \\ app\ (n+n')\ (app\ n\ \ell\ n'\ \ell')\ n''\ \ell'' &= app\ n\ \ell\ (n'+n'')\ (app\ n'\ \ell'\ n''\ \ell'') \end{aligned}$$

In the first equation, the left hand-side is of type $list(n+0)$ and the right hand-side is of type $listn$. Although one can prove that $n+0=n$ holds for any n in nat, the equality is not well-typed since $n+0$ is not $\beta\iota$-convertible to n (only terms of equivalent types can be equal).

In the second equation, the left hand-side is of type $list((n+n')+n'')$ and the right hand-side is of type $list(n+(n'+n''))$. Again, although one can prove that $(n+n')+n'' = n+(n'+n'')$ holds for any n, n' and n'' in nat, the two terms are not $\beta\iota$-convertible. Therefore, the proposition is not well-formed.

On the other hand, by adding the rules $x+0 \to x$ and $(x+y)+z \to x+(y+z)$, the previous propositions become well-typed as expected.

Integration of decision procedures. One can also define predicates by rewrite rules or having simplification rules on propositions, hence generalizing the definitions by *strong elimination* in CIC. For example, one can consider the set of rules of Figure 1 (Hsiang 1982) where $\oplus$ (exclusive "or") and $\wedge$ are commutative and associative symbols, $\bot$ represents the proposition always false and $\top$ the proposition always true.

Fig. 1. Decision procedure for classical propositional tautologies

$$\begin{aligned} P \oplus \bot &\to P \\ P \oplus P &\to \bot \\ \\ P \wedge \top &\to P \\ P \wedge \bot &\to \bot \\ P \wedge P &\to P \\ P \wedge (Q \oplus R) &\to (P \wedge Q) \oplus (P \wedge R) \end{aligned}$$

Hsiang (Hsiang 1982) showed that this system is confluent and strongly normalizing, and that a proposition P is a tautology (*i.e.* is always true) iff P reduces to $\top$. So, assuming type-checking in CC extended with this rewrite system remains decidable, then, to know whether a proposition P is a tautology, it is sufficient to submit an arbitrary proof of $\top$ to the verification program. We would not only gain in automation but also in the size of proofs (any tautology would have a proof of constant size).

We can also imagine simplification rules on equalities like the ones of Figure 2 where $+$ and $\times$ are associative and commutative, and $=$ commutative.

Fig. 2. Simplification rules on equality

$$\begin{aligned} x = x &\to \top \\ s\ x = s\ y &\to x = y \\ s\ x = 0 &\to \bot \\ x + y = 0 &\to x = 0 \wedge y = 0 \\ x \times y = 0 &\to x = 0 \vee y = 0 \end{aligned}$$

1.2. *Problems*

We saw that rewriting has numerous advantages over induction but it is not clear to which extent rewriting can be added to powerful type systems like the Calculus of Constructions (CC) without compromising the decidability of type-checking and the logical consistency. Furthermore, since rewrite rules are user-defined, it is not clear also whether $\beta\mathcal{R}$-equivalence/normalization can be made as efficient as a fixed system with $\beta\iota$-reduction only (Grégoire and Leroy 2002), although some works on rewriting seem very promising (Eker 1996; Kirchner and Moreau 2001).

Since we want to consider deductions modulo $\beta\mathcal{R}$-equivalence, we at least need this equivalence to be decidable. The usual way of proving the decidability of such an equivalence relation is by proving confluence and strong normalization of the corresponding reduction relation. Since these properties are not decidable in general, we will look for decidable sufficient conditions as general as possible.

As for the logical consistency, we cannot deduce it from normalization anymore as it is the case in CC (Barendregt 1992), since adding function symbols and rewrite rules is like adding hypothesis and equality/equivalence axioms. Therefore, for logical consistency also, we will look for sufficient conditions as general as possible.

In the following sub-section, we present a short history of the different results obtained so far on the combination of β-reduction and rewriting. Then, we will present our own contributions.

1.3. *Previous works*

The first work on the combination of typed λ-calculus and (first-order) rewriting is due to Breazu-Tannen in 1988 (Breazu-Tannen 1988). He showed that the combination of simply-typed λ-calculus and first-order rewriting is confluent if rewriting is confluent. In 1989, Breazu-Tannen and Gallier (Breazu-Tannen and Gallier 1989), and Okada (Okada 1989) independently, showed that the strong normalization also is preserved. These results were extended by Dougherty (Dougherty 1991) to any "stable" set of pure λ-terms. The combination of first-order rewriting and Pure Type Systems (PTS) (Geuvers and Nederhof 1991; Barendregt 1992) was also studied by several authors (Barbanera 1990; Barthe and Melliès 1996; Barthe and van Raamsdonk 1997; Barthe 1998).

In 1991, Jouannaud and Okada (Jouannaud and Okada 1991) extended the result of Breazu-Tannen and Gallier to the higher-order rewrite systems satisfying the *General Schema*, an extension of primitive recursion to the simply-typed λ-calculus. With higher-order rewriting, strong normalization becomes more difficult to prove since there is a strong interaction between rewriting and β-reduction, which is not the case with first-order rewriting.

In 1993, Barbanera, Fernández and Geuvers (Barbanera *et al.* 1994; Fernández 1993) extended the proof of Jouannaud and Okada to the Calculus of Constructions (CC) with object-level rewriting and simply-typed function symbols. The methods used so far for non-dependent type systems (Breazu-Tannen and Gallier 1989; Dougherty 1991) cannot be applied to dependent type systems like CC since, in this case, rewriting is included in

the type conversion rule and, thus, allows more terms to be typable. This was extended to PTS's in (Barthe and Geuvers 1995).

Other methods for proving strong normalization appeared. In 1993, Van de Pol (Van de Pol 1993; Van de Pol and Schwichtenberg 1995; Van de Pol 1996) extended to the simply-typed λ-calculus the use of monotonic interpretations. In 1999, Jouannaud and Rubio (Jouannaud and Rubio 1999) extended the Recursive Path Ordering (RPO) to the simply-typed λ-calculus.

In all these works, even the ones on CC, function symbols are always simply typed. It was Coquand (Coquand 1992) in 1992 who initiated the study of rewriting with dependent and polymorphic symbols. He studied the completeness of definitions with dependent types. He proposed a schema more general than the schema of Jouannaud and Okada since it allows inductive definitions on strictly-positive types, but it does not necessarily imply strong normalization. In 1996, Giménez (Giménez 1996; Giménez 1998) defined a restriction of this schema for which he proved strong normalization. In 1999, Jouannaud, Okada and the author (Blanqui *et al.* 2002; Blanqui *et al.* 1999) extended the General Schema in order to deal with strictly-positive types while still keeping simply-typed symbols. Finally, in 2000, Walukiewicz (Walukiewicz 2000; Walukiewicz-Chrząszcz 2002) extended Jouannaud and Rubio's HORPO to CC with dependent and polymorphic symbols.

All these works share a strong restriction: rewriting is restricted to the object level.

In 1998, Dowek, Hardin and Kirchner (Dowek *et al.* 1998) proposed a new approach to deduction for first-order logic: Natural Deduction Modulo (NDM) a congruence $\equiv$ on propositions representing the intermediate computations between two deduction steps. This deduction system consists in replacing the usual rules of Natural Deduction by equivalent rules modulo $\equiv$. For instance, the elimination rule for $\Rightarrow$ (*modus ponens*) becomes:

$$\frac{\Gamma \vdash R \quad \Gamma \vdash P}{\Gamma \vdash Q} \quad (R \equiv (P \Rightarrow Q))$$

They proved that the simple theory of types (Dowek *et al.* 2001) and skolemized set theory can be seen as first-order theories modulo congruences using *explicit substitutions* (Abadi *et al.* 1991). In (Dowek and Werner 1998; Dowek and Werner 2000), Dowek and Werner gave several conditions ensuring strong normalization of cut elimination in NDM.

1.4. *Contributions*

Our main contribution is to establish general conditions ensuring the strong normalization of the Calculus of Constructions (CC) extended with predicate-level rewriting (Blanqui 2001). In (Blanqui 2001), we show that these conditions are satisfied by most of the Calculus of Inductive Constructions (CIC) and by Natural Deduction Modulo (NDM) a large class of equational theories.

Our work can be seen as an extension of both NDM and CC, where the congruence not only includes first-order rewriting but also higher-order rewriting since, in CC, functions and predicates can be applied to functions and predicates.

It can therefore serve as a basis for a powerful extension of proof assistants like Coq (Coq Development Team 2002) or LEGO (Luo and Pollack 1992) which allow definitions by induction only. For its implementation, it may be convenient to use specialized rewriting-based applications like CiME (Contejean *et al.* 2000), ELAN (Borovanský *et al.* 2000) or Maude (Clavel *et al.* 1999). Furthermore, for program extraction (Paulin-Mohring 1989), one can imagine using rewriting-based languages and hence get more efficient extracted programs.

Considering predicate-level rewriting is not completely new. A particular case is the "strong elimination" of CIC, that is, the ability of defining predicates by induction on some inductively defined data type. The main novelty here is to consider arbitrary user-defined predicate-level rewrite rules.

Therefore, for proving the strong normalization property, we cannot completely follow the methods of Werner (Werner 1994) and Altenkirch (Altenkirch 1993) since they use in an essential way the fact that function definitions are made by induction. And the methods used in case of non-dependent first-order rewriting (Breazu-Tannen and Gallier 1989; Barbanera 1990; Dougherty 1991) cannot be applied because higher-order rewriting has a strong interaction with β-reduction and because, in dependent type systems, rewriting allows more terms to be typable. Our method is based on the notion of reducibility candidates of Tait and Girard (Girard *et al.* 1988) and extend Geuvers' method (Geuvers 1994) for dealing with rewriting.

Let us mention two other important contributions.

For allowing some quotient types (rules on constructors) and matching on function symbols, which is not possible in CIC, we use a notion of constructor more general than the usual one (see Section 5.1).

For ensuring the subject reduction property, that is, the preservation of typing under reduction, we introduce conditions more general than the ones used so far. In particular, these conditions allow us to get rid of non-linearities due to typing, which makes rewriting more efficient and confluence easier to prove (see Section 3).

2. The Calculus of Algebraic Constructions

The Calculus of Algebraic Constructions (CAC) is an extension of the Calculus of Constructions (CC) (Coquand and Huet 1988) with function and predicate symbols defined by rewrite rules.

2.1. *Terms*

CC is a particular Pure Type System (PTS) (Barendregt 1992) defined from a set $\mathcal{S} = \{\star, \Box\}$ of *sorts*. The sort $\star$ is intended to be the type of data types and propositions, while the sort $\Box$ is intended to be the type of predicate types (also called *kinds*). For instance, the type nat of natural numbers is of type $\star$, $\star$ is of type $\Box$, the predicate $\leq$ on natural numbers is of type $nat \Rightarrow nat \Rightarrow \star$, and $nat \Rightarrow nat \Rightarrow \star$ is of type $\Box$.

The terms of CC are usually defined by the following grammar rule:

$$t ::= s \mid x \mid [x : t]t \mid (x : t)t \mid tt$$

where s is a sort, x a variable, $[x : t]t$ an abstraction, $(x : t)t$ a (dependent) product, and tt an application. We assume that the set $\mathcal{X}$ of variables is an infinite denumerable set disjoint from $\mathcal{S}$.

We simply extend CC by considering a denumerable set $\mathcal{F}$ of *symbols*, disjoint from $\mathcal{S}$ and $\mathcal{X}$, and by adding the following new construction:

$$t ::= \ldots \mid f \in \mathcal{F}$$

We denote by $\mathcal{T}(\mathcal{F}, \mathcal{X})$ the set of terms built from $\mathcal{F}$ and $\mathcal{X}$. Note that, in contrast with (Blanqui 2001), function symbols are curried. No notion of arity is required.

2.2. *Notations*

Free and bound variables. A variable x in the scope of an abstraction $[x : T]$ or a product $(x : T)$ is *bound*. As usual, it may be replaced by any other variable. This is *α-equivalence*. A variable which is not bound is *free*. We denote by $\mathrm{FV}(t)$ the set of free variables of a term t. A term without free variable is *closed*. We often denote by $U \Rightarrow V$ a product $(x : U)V$ with $x \notin \mathrm{FV}(V)$ (non-dependent product). See (Barendregt 1992) for more details on these notions.

Vectors. We often use vectors $(\vec{t}, \vec{u}, \ldots)$ for sequences of terms (or anything else). The size of a vector $\vec{t}$ is denoted by $|\vec{t}|$. For instance, $[\vec{x} : \vec{T}]u$ denotes the term $[x_1 : T_1] \ldots [x_n : T_n]u$ where $n = |\vec{x}|$.

Positions. To designate a subterm of a term, we use a system of *positions à la* Dewey (words over the alphabet of positive integers). Formally, the set $\mathrm{Pos}(t)$ of the positions in a term t is inductively defined as follows:

- $\mathrm{Pos}(f) = \mathrm{Pos}(s) = \mathrm{Pos}(x) = \{\varepsilon\}$,
- $\mathrm{Pos}((x : t)u) = \mathrm{Pos}([x : t]u) = \mathrm{Pos}(tu) = 1.\mathrm{Pos}(t) \cup 2.\mathrm{Pos}(u)$,

where ε denotes the empty word and '.' the concatenation. We denote by $t|_p$ the subterm of t at the position p, and by $t[u]_p$ the term obtained by replacing $t|_p$ by u in t. The relation "is a subterm of" is denoted by $\trianglelefteq$, and its strict part by $\triangleleft$.

We denote by $\mathrm{Pos}(f, t)$ the set of positions p in t such that $t|_p = f$, and by $\mathrm{Pos}(x, t)$ the set of positions p in t such that $t|_p$ is a free occurrence of x in t.

Substitutions. A *substitution* θ is an application from $\mathcal{X}$ to $\mathcal{T}$ whose *domain* $\mathrm{dom}(\theta) = \{x \in \mathcal{X} \mid x\theta \neq x\}$ is finite. Its set of free variables is $\mathrm{FV}(\theta) = \bigcup\{\mathrm{FV}(x\theta) \mid x \in \mathrm{dom}(\theta)\}$. Applying a substitution θ to a term t consists of replacing every variable x free in t by its image $x\theta$ (to avoid variable captures, bound variables must be distinct from free variables). The result is denoted by $t\theta$. We denote by $\{\vec{x} \mapsto \vec{t}\}$ the substitution which associates t_i to x_i, and by $\theta \cup \{x \mapsto t\}$ the substitution which associates t to x and $y\theta$ to $y \neq x$.

Relations. Let $\to$ be a relation on terms. We denote by:

– $\rightarrow(t)$ the set of terms t' such that $t \rightarrow t'$,
– $\leftarrow$ the inverse of $\rightarrow$,
– $\rightarrow^+$ the smallest transitive relation containing $\rightarrow$,
– $\rightarrow^*$ the smallest reflexive and transitive relation containing $\rightarrow$,
– $\leftrightarrow^*$ the smallest reflexive, transitive and symmetric relation containing $\rightarrow$,
– $\downarrow$ the relation $\rightarrow^* {}^*\!\leftarrow$ ($t \downarrow u$ if there exists v such that $t \rightarrow^* v$ and $u \rightarrow^* v$).

If $t \rightarrow t'$ then we say that t *rewrites* to t'. If $t \rightarrow^* t'$ then we say that t *reduces* to t'. A relation $\rightarrow$ is *stable by context* if $u \rightarrow u'$ implies $t[u]_p \rightarrow t[u']_p$ for all term t and position $p \in \mathrm{Pos}(t)$. The relation $\rightarrow$ is *stable by substitution* if $t \rightarrow t'$ implies $t\theta \rightarrow t'\theta$ for all substitution θ.

The *β-reduction* (resp. *η-reduction*) relation is the smallest relation stable by context and substitution containing $[x : U]v\ u \rightarrow_\beta v\{x \mapsto u\}$ (resp. $[x : U]tx \rightarrow_\eta t$ if $x \notin \mathrm{FV}(t)$). A term of the form $[x : U]v\ u$ (resp. $[x : U]tx$ with $x \notin \mathrm{FV}(t)$) is a *β-redex* (resp. *η-redex*).

A relation $\rightarrow$ is *weakly normalizing* if, for all term t, there exists an irreducible term t' to which t reduces. We say that t' is a *normal form* of t. A relation $\rightarrow$ is *strongly normalizing* (well-founded, nœtherian) if, for all term t, any reduction sequence issued from t is finite.

The relation $\rightarrow$ is *locally confluent* if, whenever a term t rewrites to two distinct terms u and v, then $u \downarrow v$. The relation $\rightarrow$ is *confluent* if, whenever a term t reduces to two distinct terms u and v, then $u \downarrow v$.

If $\rightarrow$ is locally confluent and strongly normalizing then $\rightarrow$ is confluent (Newman's lemma). If $\rightarrow$ is confluent and weakly normalizing then every term t has a unique normal form denoted by $t\downarrow$.

Orderings. A *precedence* is a quasi-ordering on $\mathcal{F}$ whose strict part is well-founded. Let $>_1, \ldots, >_n$ be orderings on the sets $E_1, \ldots, E_n$ respectively. We denote by $(>_1, \ldots, >_n)_{\mathrm{lex}}$ the *lexicographic* ordering on $E_1 \times \ldots \times E_n$. Now, let $>$ be an ordering on a set E. We denote by $>_{\mathrm{mul}}$ the ordering on finite multisets on E. An important property of these extensions is that they preserve well-foundedness. See (Baader and Nipkow 1998) for more details on these notions.

2.3. *Rewriting*

In first-order frameworks, that is, in a first-order term algebra, a rewrite rule is generally defined as a pair $l \rightarrow r$ of terms such that l is not a variable and the variables occurring in r also occur in l (otherwise, rewriting does not terminate). Then, one says that a term t rewrites to a term t' at position p, written $t \rightarrow^p t'$, if there exists a substitution σ such that $t|_p = l\sigma$ and $t' = t[r\sigma]_p$. See (Dershowitz and Jouannaud 1990) for more details on (first-order) rewriting.

Here, we consider a very similar rewriting mechanism by restricting left-hand sides of rules to be algebraic. On the other hand, right-hand sides can be arbitrary. This is a particular case of *Combinatory Reduction System* (CRS) (Klop *et al.* 1993) for which it is not necessary to use *higher-order pattern-matching*. However, we proved in (Blanqui

2000) that, in case of simply-typed λ-calculus, our termination criteria can be adapted to rewriting with higher-order pattern-matching.

Definition 1 (Rewriting) Terms only built from variables and applications of the form $f\vec{t}$ with $f \in \mathcal{F}$ are said *algebraic*. A *rewrite rule* is a pair of terms $l \to r$ such that l is algebraic, distinct from a variable and $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$. A rule $l \to r$ is *left-linear* if no variable occurs more than once in l. A rule $l \to r$ is *non-duplicating* if no variable has more occurences in r than in l. A rule $f\vec{l} \to r$ is *compatible with a precedence* $\geq$ if, for all symbol g occuring in r, $f \geq g$.

Let $\mathcal{R}$ be a denumerable set of rewrite rules. The *$\mathcal{R}$-reduction relation* $\to_{\mathcal{R}}$ is the smallest relation containing $\mathcal{R}$ and stable by substitution and context. A term of the form $l\sigma$ with $l \to r \in \mathcal{R}$ is an *$\mathcal{R}$-redex*. We assume that $\to_{\mathcal{R}}$ is finitely branching.

Given a set $\mathcal{G} \subseteq \mathcal{F}$, we denote by $\mathcal{R}_{\mathcal{G}}$ the set of rules that *define* a symbol in $\mathcal{G}$, that is, whose left-hand side is headed by a symbol in $\mathcal{G}$. A symbol f is *constant* if $\mathcal{R}_{\{f\}} = \emptyset$, otherwise it is (partially) *defined*. We denote by $\mathcal{CF}$ the set of constant symbols and by $\mathcal{DF}$ the set of defined symbols.

2.4. *Typing*

We now define the set of *well-typed* terms. An *environment* Γ is a list of pairs $x : T$ made of a variable x and a term T. We denote by $\emptyset$ the empty environment and by $\mathcal{E}(\mathcal{F}, \mathcal{X})$ the set of environments built from $\mathcal{F}$ and $\mathcal{X}$. The *domain* of an environment Γ, $\mathrm{dom}(\Gamma)$, is the set of variables x such that a pair $x : T$ belongs to Γ. If $x \in \mathrm{dom}(\Gamma)$ then we denote by $x\Gamma$ the first term T such that $x : T$ belongs to Γ. The set of *free variables* in an environment Γ is $\mathrm{FV}(\Gamma) = \bigcup\{\mathrm{FV}(x\Gamma) \mid x \in \mathrm{dom}(\Gamma)\}$. Given two environments Γ and Γ', Γ is *included* in Γ', written $\Gamma \subseteq \Gamma'$, if all the elements of Γ occur in Γ' in the same order.

Definition 2 (Typing) We assume that every variable x is equipped with a sort s_x, that the set $\mathcal{X}^s$ of variables of sort s is infinite, and that α-equivalence preserves sorts. Let $\mathrm{FV}^s(t) = \mathrm{FV}(t) \cap \mathcal{X}^s$ and $\mathrm{dom}^s(\Gamma) = \mathrm{dom}(\Gamma) \cap \mathcal{X}^s$. We also assume that every symbol f is equipped with a sort s_f and a closed type $\tau_f = (\vec{x} : \vec{T})U$ such that, for all rule $f\vec{l} \to r$, $|\vec{l}| \leq |\vec{x}|$. We often write $f : T$ for saying that $\tau_f = T$.

The typing relation of a CAC is the smallest ternary relation $\vdash \subseteq \mathcal{E} \times \mathcal{T} \times \mathcal{T}$ defined by the inference rules of Figure 3 where $s, s' \in \mathcal{S}$. A term t is *typable* if there exists an environment Γ and a term T such that $\Gamma \vdash t : T$ (T is a *type* of t in Γ). In the following, we always assume that $\vdash \tau_f : s_f$ for all $f \in \mathcal{F}$.

An environment is *valid* if a term is typable in it. A substitution θ is *well-typed from* Γ *to* Δ, $\theta : \Gamma \rightsquigarrow \Delta$, if, for all $x \in \mathrm{dom}(\Gamma)$, $\Delta \vdash x\theta : x\Gamma\theta$. We denote by $T\ \mathcal{C}_\Gamma\ T'$ the fact that $T \downarrow T'$ and $\Gamma \vdash T' : s'$, and by $T\ \mathbb{C}_\Gamma\ T'$ the fact that $T\ \mathcal{C}_\Gamma\ T'$ and $\Gamma \vdash T : s$.

Compared with CC, we have a new rule, (symb), for typing symbols and, in the type conversion rule (conv), we have $\downarrow_{\beta\mathcal{R}}$ (that we simply denote by $\downarrow$ in the rest of the paper) instead of the β-conversion $\leftrightarrow^*_\beta = \downarrow_\beta$ (since β is confluent).

Fig. 3. Typing rules

$$\text{(ax)}\quad \frac{}{\vdash \star : \Box}$$

$$\text{(symb)}\quad \frac{\vdash \tau_f : s_f}{\vdash f : \tau_f}$$

$$\text{(var)}\quad \frac{\Gamma \vdash T : s_x}{\Gamma, x : T \vdash x : T} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{(weak)}\quad \frac{\Gamma \vdash t : T \quad \Gamma \vdash U : s_x}{\Gamma, x : U \vdash t : T} \qquad (x \notin \mathrm{dom}(\Gamma))$$

$$\text{(prod)}\quad \frac{\Gamma \vdash U : s \quad \Gamma, x : U \vdash V : s'}{\Gamma \vdash (x : U)V : s'}$$

$$\text{(abs)}\quad \frac{\Gamma, x : U \vdash v : V \quad \Gamma \vdash (x : U)V : s}{\Gamma \vdash [x : U]v : (x : U)V}$$

$$\text{(app)}\quad \frac{\Gamma \vdash t : (x : U)V \quad \Gamma \vdash u : U}{\Gamma \vdash tu : V\{x \mapsto u\}}$$

$$\text{(conv)}\quad \frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s'}{\Gamma \vdash t : T'} \qquad (T \downarrow_{\beta\mathcal{R}} T')$$

Well-typed substitutions enjoy the following important substitution property: if $\Gamma \vdash t : T$ and $\theta : \Gamma \rightsquigarrow \Delta$ then $\Delta \vdash t\theta : T\theta$.

The relations $\mathcal{C}_\Gamma$ (not symmetric) and $\mathbb{C}_\Gamma$ (symmetric) are useful when inverting typing judgements. For instance, a derivation of $\Gamma \vdash uv : W'$ necessarily terminates by an application of the (app) rule, possibly followed by applications of the rules (weak) and (conv). Therefore, there exists V and W such that $\Gamma \vdash u : (x : V)W$, $\Gamma \vdash v : V$ and $W\{x \mapsto v\}\ \mathcal{C}_\Gamma^*\ W'$. Since, in the (conv) rule, T is not required to be typable by some sort s (as it is the case for T'), it is not *a priori* the case that $W\{x \mapsto v\}$ is typable and therefore that, in fact, $W\{x \mapsto v\}\ \mathbb{C}_\Gamma^*\ W'$.

Many of the well-known basic properties of Pure Type Systems (PTS's) (Barendregt 1992) also hold for CAC's. In (Blanqui 2001), we study these properties in an abstract way by considering a PTS equipped with an unspecified type conversion rule (instead of $\downarrow_\beta$ or $\downarrow_{\beta\mathcal{R}}$ for instance), hence factorizing several previous proofs for different PTS extensions. The properties we use in this paper are:

(type correctness) If $\Gamma \vdash t : T$ then either $T = \Box$ or $\Gamma \vdash T : s$.
(conversion correctness) If $\Gamma \vdash T : s$ and $T\ \mathbb{C}_\Gamma^*\ T'$ then $\Gamma \vdash T' : s$.
(convertibility of types) If $\Gamma \vdash t : T$ and $\Gamma \vdash t : T'$ then $T\ \mathbb{C}_\Gamma^*\ T'$.

Only convertibility of types requires confluence (conversion correctness is proved in Section 3.2 without using confluence).

Among well-typed terms, we distinguish:
- The set $\mathbb{K}$ of *predicate types* or *kinds* made of the terms K such that $\Gamma \vdash K : \Box$. It is easy to check that every predicate type is of the form $(\vec{x} : \vec{T})\star$.
- The set $\mathbb{P}$ of *predicates* made of the terms T such that $\Gamma \vdash T : K$ and $\Gamma \vdash K : \Box$.
- The set $\mathbb{O}$ of *objects* made of the terms t such that $\Gamma \vdash t : T$ and $\Gamma \vdash T : \star$.

3. Subject reduction

Before studying the strong normalization or the logical consistency of our system, we must make sure that the reduction relation $\rightarrow_{\beta\mathcal{R}}$ is indeed correct w.r.t. typing, that is, if $\Gamma \vdash t : T$ and $t \rightarrow_{\beta\mathcal{R}} t'$ then $\Gamma \vdash t' : T$. This property is usually called *subject reduction*. Once it holds, it can be easily extended to types, environments and substitutions:
- If $\Gamma \vdash t : T$ and $T \rightarrow T'$ then $\Gamma \vdash t : T'$.
- If $\Gamma \vdash t : T$ and $\Gamma \rightarrow \Gamma'$ then $\Gamma' \vdash t : T$.
- If $\theta : \Gamma \rightsquigarrow \Delta$ and $\theta \rightarrow \theta'$ then $\theta' : \Gamma \rightsquigarrow \Delta$.

In presence of dependent types and rewriting, the subject reduction for β appears to be a difficult problem. Indeed, in the case of a head-reduction $[x : U']v\ u \rightarrow_\beta v\{x \mapsto u\}$ with $\Gamma \vdash [x : U']v : (x : U)V$ and $\Gamma \vdash u : U$, we must prove that $\Gamma \vdash v\{x \mapsto u\} : V\{x \mapsto u\}$. By inversion, we have $\Gamma, x : U' \vdash v : V'$ with $(x : U')V'\ \mathbb{C}^*_\Gamma\ (x : U)V$. We can conclude that $\Gamma \vdash v\{x \mapsto u\} : V\{x \mapsto u\}$ only if:

$$(x : U')V'\ \mathbb{C}^*_\Gamma\ (x : U)V \text{ implies } U'\ \mathbb{C}^*_\Gamma\ U \text{ and } V'\ \mathbb{C}^*_{\Gamma,x:U}\ V,$$

a property that we call *product compatibility*.

This is immediate as soon as $\rightarrow_{\beta\mathcal{R}}$ is confluent. Unfortunately, there are very few results on the confluence of higher-order rewriting and β-reduction together (see the discussion after Definition 29). Fortunately, confluence is not the only way to prove the product compatibility. In (Geuvers 1993), Geuvers proves the product compatibility for the Calculus of Constructions (CC) with $\leftrightarrow^*_{\beta\eta}$ as type conversion relation, although $\rightarrow_{\beta\eta}$ is not confluent on untyped terms: $[x : T]x\ {}_\beta\!\leftarrow [x : T]([y : U]y\ x) \rightarrow_\eta [y : U]y =_\alpha [x : U]x$ (Nederpelt 1973). And, in (Barbanera *et al.* 1997), Barbanera, Geuvers and Fernández prove the product compatibility for CC with $\downarrow_\beta \cup \downarrow_{\mathcal{R}}$ as type conversion relation, where $\mathcal{R}$ is a set of simply-typed object-level rewrite rules.

In Section 3.2, we prove the product compatibility, hence the subject reduction of β, for a large class of rewrite systems, including predicate-level rewriting, without using confluence, by generalizing the proof of Barbanera, Fernández and Geuvers (Barbanera *et al.* 1997). Before that, we study the subject reduction for rewriting.

3.1. *Subject reduction for rewriting*

In first-order sorted algebras, for rewriting to preserve sorts, it suffices that both sides of a rule have the same sort. Carried over to type systems, this condition gives: there exists

an environment Γ and a type T such that $\Gamma \vdash l : T$ and $\Gamma \vdash r : T$. This condition is the one which has been taken in all previous work combining typed λ-calculus and rewriting. However, it has an important drawback. With polymorphic or dependent types, it leads to strongly non left-linear rules, which has two important consequences. First, rewriting is strongly slowed down because of the necessary equality tests. Second, it is more difficult to prove confluence.

Let us take the example of the concatenation of two polymorphic lists (type $list : \star \Rightarrow \star$ with the constructors $nil : (A : \star)listA$ and $cons : (A : \star)A \Rightarrow listA \Rightarrow listA$):

$$\begin{array}{rcl} app\ A\ (nil\ A)\ \ell' & \to & \ell' \\ app\ A\ (cons\ A\ x\ \ell)\ \ell' & \to & cons\ A\ x\ (app\ A\ \ell\ \ell') \end{array}$$

This definition satisfies the usual condition by taking $\Gamma = A : \star, x : A, \ell : listA, \ell' : listA$ and $T = listA$. But one may wonder whether it is really necessary to do an equality test between the first argument of *app* and the first argument of *cons* when one wants to apply the second rule. Indeed, if $app\ A\ (cons\ A'\ x\ \ell)\ \ell'$ is well-typed then, by inversion, $cons\ A'\ x\ \ell$ is of type $listA$ and, by inversion again, $listA'$ is convertible to $listA$. Thus, A is convertible to A'.

In fact, what is important is not that the left-hand side of a rule be typable, but that, if an instance of the left-hand side of a rule is typable, then the corresponding instance of the right-hand side has the same type. We express this by requiring that there exists an environment Γ in which the right-hand side is typable, and a substitution ρ which replaces the variables of the left-hand side not belonging to Γ by terms typable in Γ. Hence, one can consider the following rules instead:

$$\begin{array}{rcl} app\ A\ (nil\ A')\ \ell' & \to & \ell' \\ app\ A\ (cons\ A'\ x\ \ell)\ \ell' & \to & cons\ A\ x\ (app\ A\ \ell\ \ell') \end{array}$$

by taking $\Gamma = A : \star, x : A, \ell : listA, \ell' : listA$ and $\rho = \{A' \mapsto A\}$.

Definition 3 (Well-typed rule) A rule $l \to r$ with $l = f\vec{l}$, $f : (\vec{x} : \vec{T})U$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$ is *well-typed* if there exists an environment Γ and a substitution ρ such that:[‖]

(S3) $\Gamma \vdash r : U\gamma\rho$,

(S4) $\forall \Delta, \sigma, T$, if $\Delta \vdash l\sigma : T$ then $\sigma : \Gamma \rightsquigarrow \Delta$,

(S5) $\forall \Delta, \sigma, T$, if $\Delta \vdash l\sigma : T$ then $\sigma \downarrow \rho\sigma$.

In the following, we write $(l \to r, \Gamma, \rho) \in \mathcal{R}$ when the previous conditions are satisfied.

An example with dependent types is given by the concatenation of two lists of fixed length (type $list : nat \Rightarrow \star$ with the constructors $nil : list\,0$ and $cons : nat \Rightarrow (n : nat)\ list\,n \Rightarrow list\,(s\,n)$) and the function *map* which applies a function f to every element of a list:

$$\begin{array}{l} app : (n : nat)list\,n \Rightarrow (n' : nat)list\,n' \Rightarrow list\,(n + n') \\ map : (nat \Rightarrow nat) \Rightarrow (n : nat)list\,n \Rightarrow list\,n \end{array}$$

[‖] The conditions **(S1)** $\mathrm{dom}(\rho) \cap \mathrm{dom}(\Gamma) = \emptyset$ and **(S2)** $\Gamma \vdash l\rho : U\gamma\rho$ given in (Blanqui 2001) are not necessary for proving the subject reduction property, but they are necessary for proving the strong normalization property of the higher-order rewrite rules (see Definition 26).

where *app* and *map* are defined by:

$$\begin{array}{rcl} app\ 0\ \ell\ n'\ \ell' & \to & \ell' \\ app\ p\ (cons\ x\ n\ \ell)\ n'\ \ell' & \to & cons\ x\ (n+n')\ (app\ n\ \ell\ n'\ \ell') \\ map\ f\ 0\ \ell & \to & \ell \\ map\ f\ p\ (cons\ x\ n\ \ell) & \to & cons\ (f\ x)\ n\ (map\ f\ n\ \ell) \\ map\ f\ p\ (app\ n\ \ell\ n'\ \ell') & \to & app\ n\ (map\ f\ n\ \ell)\ n'\ (map\ f\ n'\ \ell') \end{array}$$

For the second rule of *app*, we take $\Gamma = x : nat, n : nat, \ell : list\, n, n' : nat, \ell' : list\, n'$ and $\rho = \{p \mapsto sn\}$. This avoids checking that p is convertible to sn. For the third rule of *map*, we take $\Gamma = f : nat \Rightarrow nat, n : nat, \ell : list\, n, n' : nat, \ell' : list\, n'$ and $\rho = \{p \mapsto n + n'\}$. This avoids checking that p is convertible to $n + n'$. The reader will find more examples at the end of Section 5.

Lemma 4 If $\beta\mathcal{R}$ is product compatible, $f : (\vec{x} : \vec{T})U$, $\theta = \{\vec{x} \mapsto \vec{t}\}$ and $\Gamma \vdash f\vec{t} : T$ then $\theta : \Gamma_f \rightsquigarrow \Gamma$ and $U\theta\ \mathbb{C}^*_\Gamma\ T$.

Proof. By inversion, there is a sequence of products $(x_i : T'_i)U_i$ $(1 \leq i \leq n = |\vec{x}|)$ such that $\Gamma \vdash ft_1 \ldots t_{n-1} : (x_n : T'_n)U_n$, $\Gamma \vdash t_n : T'_n$, $U_n\theta\ \mathbb{C}^*_\Gamma\ T$, ..., $\Gamma \vdash f : (x_1 : T'_1)U_1$, $\Gamma \vdash t_1 : T'_1$, $U_1\theta\ \mathbb{C}^*_\Gamma\ (x_2 : T'_2)U_2$ and $(\vec{x} : \vec{T})U\ \mathbb{C}^*_\Gamma\ (x_1 : T'_1)U_1$. Let $V_i = (x_{i+1} : T_{i+1}) \ldots (x_n : T_n)U$. By product compatibility, $T_1\theta = T_1\ \mathbb{C}^*_\Gamma\ T'_1$ and $V_1\ \mathbb{C}^*_{\Gamma, x_1:T_1}\ U_1$. Hence, $V_1\theta = (x_2 : T_2\theta)V_2\theta\ \mathbb{C}^*_\Gamma\ U_1\theta\ \mathbb{C}^*_\Gamma\ (x_2 : T'_2)U_2$. Therefore, by induction, $T_2\theta\ \mathbb{C}^*_\Gamma\ T'_2$, ..., $T_n\theta\ \mathbb{C}^*_\Gamma\ T'_n$ and $U\theta\ \mathbb{C}^*_\Gamma\ U_n\theta\ \mathbb{C}^*_\Gamma\ T$. Hence, by conversion, $\Gamma \vdash t_i : T_i\theta$, that is, $\theta : \Gamma_f \rightsquigarrow \Gamma$. □

Theorem 5 (Subject reduction for $\mathcal{R}$) If $\beta\mathcal{R}$ is product compatible and $\mathcal{R}$ is a set of well-typed rules then $\mathcal{R}$ preserves typing.

Proof. As usual, we prove by induction on $\Delta \vdash t : T$ that, if $t \to_\mathcal{R} t'$ then $\Delta \vdash t' : T$, and if $\Delta \to_\mathcal{R} \Delta'$ then $\Delta' \vdash t : T$. We only detail the (app) case. Assume that $\Delta \vdash l\sigma : T$, $(l \to r, \Gamma, \rho) \in \mathcal{R}$, $l = f\vec{l}$, $f : (\vec{x} : \vec{T})U$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$. Let $\theta = \gamma\sigma$. After Lemma 4, $\theta : \Gamma_f \rightsquigarrow \Delta$ and $U\theta\ \mathbb{C}^*_\Delta\ T$. By **(S4)**, $\sigma : \Gamma \rightsquigarrow \Delta$. By **(S3)**, $\Gamma \vdash r : U\gamma\rho$. Therefore, by substitution, $\Delta \vdash r\sigma : U\gamma\rho\sigma$. By **(S5)**, $\rho\sigma \downarrow \sigma$. Therefore, by conversion, $\Delta \vdash r\sigma : U\theta$ and $\Delta \vdash r\sigma : T$. □

How to check the conditions (S3), (S4) and (S5) ? In all their generality, they are certainly undecidable. On the one hand, we do not know whether $\vdash$ and $\downarrow$ are decidable and, on the other hand, in (S4) and (S5), we arbitrarily quantify over Δ, σ and T. It is therefore necessary to put additional restrictions. In the following, we successively consider the three conditions.

Let us look at (S3). In practice, the symbols and their defining rules are often added one after another (or by groups but the following argument can be generalized). Let $(\mathcal{F}, \mathcal{R})$ be a system in which $\vdash$ is decidable, $f \notin \mathcal{F}$ and $\mathcal{R}_f$ a set of rules defining f and whose symbols belong to $\mathcal{F}' = \mathcal{F} \cup \{f\}$. Then, in $(\mathcal{F}', \mathcal{R})$, $\vdash$ is still decidable. One can therefore try to check (S3) in this system. This does not seem an important restriction: it would be surprising if the typing of a rule requires the use of the rule itself !

We now consider (S4).

Definition 6 (Canonical and derived types) Let t be a term of the form $l\sigma$ with $l = f\vec{l}$ algebraic, $f : (\vec{x} : \vec{T})U$, $n = |\vec{x}| = |\vec{l}|$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$. The term $U\gamma\sigma$ will be called the *canonical type* of t. Let $p \in \mathrm{Pos}(l)$ of the form $(1^*2)^+$. We inductively define the *type of* $t|_p$ *derived from* t, $\tau(t,p)$, as follows:

- if $p = 1^{n-i}2$ then $\tau(t,p) = T_i\gamma\sigma$,
- if $p = 1^{n-i}2q$ and $q \neq \varepsilon$ then $\tau(t,p) = \tau(t_i,q)$.

The type of $t|_p$ derived from t only depends on the term above $t|_p$.

Lemma 7 (S4) If, for all $x \in \mathrm{dom}(\Gamma)$, there is $p \in \mathrm{Pos}(x,l)$ such that $x\Gamma = \tau(l,p)$, then (S4) is satisfied.

Proof. We prove (S4) by induction on the size of l. Assume that $\Delta \vdash l\sigma : T$. We must prove that, for all $x \in \mathrm{dom}(\Gamma)$, $\Delta \vdash x\sigma : x\Gamma\sigma$. By assumption, there is $p \in \mathrm{Pos}(x,l)$ such that $x\Gamma = \tau(l,p)$. Since $l = f\vec{l}$, $p = jq$. Assume that $f : (\vec{x} : \vec{T})U$. Let $\gamma = \{\vec{x} \mapsto \vec{l}\}$ and $\theta = \gamma\sigma$. If $q = \varepsilon$ then $x = l_j$ and $x\Gamma = T_j\gamma$. Now, after Lemma 4, $\theta : \Gamma_f \rightsquigarrow \Delta$. So, $\Delta \vdash x_j\theta : T_j\theta$, that is, $\Delta \vdash x\sigma : x\Gamma\sigma$. Assume now that $q \neq \varepsilon$. Since $\Delta \vdash l_j\sigma : T_j\theta$, l_j is of the form $g\vec{m}$ and $x\Gamma = \tau(l_j,q)$, by induction hypothesis, $\Delta \vdash x\sigma : x\Gamma\sigma$. □

For (S5), we have no general result. By inversion, (S5) can be seen as a unification problem modulo $\downarrow^*$. The confluence of $\rightarrow$ (which implies that $\downarrow^*=\downarrow$) can therefore be very useful. Unfortunately, there are very few results on the confluence of the combination of higher-order rewriting and β-reduction (see the discussion after Definition 29). On the other hand, one can easily prove that local confluence is preserved.

Theorem 8 (Local confluence) If $\mathcal{R}$ is locally confluent on algebraic terms then $\beta\mathcal{R}$ is locally confluent on any term.

Proof. Assume that $t \rightarrow^p t_1$ and $t \rightarrow^q t_2$. We prove by induction on t that there exists t' such that $t_1 \rightarrow^* t'$ and $t_2 \rightarrow^* t'$. There are three cases:

- $p \sharp q$ (p and q have no common prefix). The reductions at p and q can be done in parallel: $t_1 \rightarrow^q t_1'$, $t_2 \rightarrow^p t_2'$ and $t_1' = t_2'$.
- $p = ip'$ and $q = iq'$. We can conclude by induction hypothesis on $t|_i$.
- $p = \varepsilon$ or $q = \varepsilon$. By exchanging the roles of p and q, we can assume that $p = \varepsilon$. There are two cases:
  - $t = [x : V]u\ v$ and $t_1 = u\{x \mapsto v\}$. We distinguish three sub-cases:
    - $q = 11q'$ and $V \rightarrow^{q'} V'$. Then, $t' = t_1$ works.
    - $q = 12q'$ and $u \rightarrow^{q'} u'$. Then, $t' = u'\{x \mapsto v\}$ works.
    - $q = 2q'$ and $v \rightarrow^{q'} v'$. Then, $t' = u\{x \mapsto v'\}$ works.
  - $t = l\sigma$, $l \rightarrow r \in \mathcal{R}$ and $t_1 = r\sigma$. There exists an algebraic term u of maximal size and a substitution θ such that $t = u\theta$ and $x\theta = y\theta$ implies $x = y$ (u and θ are unique up to the choice of variables and u has the same non-linearities than t). As the left-hand sides of rules are algebraic, $u = l\sigma'$ and $\sigma = \sigma'\theta$. Now, we distinguish two sub-cases:
    - $q \in \mathrm{Pos}(u)$. As the left-hand sides of rules are algebraic, we have $u \rightarrow_{\mathcal{R}} r\sigma'$ and

$u \to_{\mathcal{R}} v$. By local confluence of $\to_{\mathcal{R}}$ on algebraic terms, there exists u' such that $r\sigma' \to^* u'$ and $v \to^* u'$. Then, $t' = u'\theta$ works.

- $q = q_1 q'$ and $u|_{q_1} = x$. Let $q_2, \ldots, q_n$ be the positions of the other occurrences of x in u. If one reduces t_2 at each position $q_i q'$, one obtains a term of the form $l\sigma'\theta'$ where θ' is the substitution such that $x\theta'$ is the reduct of $x\theta$, and $y\theta' = y\theta$ if $y \neq x$. Then, $t' = r\sigma'\theta'$ works.

□

3.2. *Subject reduction for β*

In this section, we prove the product compatibility, hence the subject reduction of β, for a large class of rewrite systems, including predicate-level rewrite rules, without using confluence, by generalizing the proof of Barbanera, Fernández and Geuvers (Barbanera *et al.* 1997). It is worth noting that no result of this section assumes the subject reduction property for rewriting. They only rely on simple syntactic properties of β-reduction and rewriting with respect to predicates and kinds (Lemma 11).

The idea is to β-weak-head normalize all the intermediate terms between $(x : U')V'$ and $(x : U)V$ so that we obtain a sequence of conversions between product terms only. We first show that the subject reduction property can indeed be studied in a system whose conversion relation is like the one used in (Barbanera *et al.* 1997).

Lemma 9 Let Λ be a CAC with conversion relation $\downarrow_{\beta\mathcal{R}}$ and Λ' be the same CAC but with conversion relation $\downarrow_\beta \cup \downarrow_{\mathcal{R}}$. If $\to_{\beta\mathcal{R}}$ has the subject reduction property in Λ' then $\Lambda = \Lambda'$ (and $\to_{\beta\mathcal{R}}$ has the subject reduction property in Λ).

Proof. Let $\vdash$ (resp. $\vdash'$) be the typing relation of Λ (resp. Λ'). Since $\downarrow_\beta \cup \downarrow_{\mathcal{R}} \subseteq \downarrow_{\beta\mathcal{R}}$, we clearly have $\vdash' \subseteq \vdash$. We prove by induction on $\vdash$ that $\vdash \subseteq \vdash'$. The only difficult case is of course (conv). By induction hypothesis, we have $\Gamma \vdash' t : T$ and $\Gamma \vdash' T' : s'$. Furthermore, we have $T \to^*_{r_1} \to^*_{r_2} \ldots {}^*_{s_2} \leftarrow {}^*_{s_1} \leftarrow T'$ with $r_k, s_k \in \{\beta, \mathcal{R}\}$. By type correctness, either $T = \Box$ or there is a sort s such that $\Gamma \vdash' T : s$. If $T = \Box$ then $T' \to^* \Box$. But, since $\to$ has the subject reduction property in Λ', we get that $\Gamma \vdash' \Box : s'$, which is not possible. Therefore, T and T' are typable in Λ' and, since $\to$ has the subject reduction property in Λ', all the terms between T and T' are also typable in Λ'. Therefore, we can replace the conversion in Λ by a sequence of conversions in Λ'. □

We now prove a series of useful results about kinds and predicates which will allow us to prove the subject reduction property on types for the β-weak-head reduction relation h: $t \to_h t'$ if $t = [\vec{x} : \vec{T}]([x : U]v u\vec{t})$ and $t' = [\vec{x} : \vec{T}](v\{x \mapsto u\}\vec{t})$. The β-internal reduction relation will be denoted by $\not\hbar$. To this end, we introduce several sets of terms.

- $\mathcal{K}$: terms of the form $(\vec{x} : \vec{T})\star$, usually called *kinds*.
- $\mathcal{P}$: smallest set of terms, called *predicates*, such that $\mathcal{X}^\Box \cup \mathcal{F}^\Box \subseteq \mathcal{P}$ and, if $pt \in \mathcal{P}$ or $[x : t]p \in \mathcal{P}$ or $(x : t)p \in \mathcal{P}$, then $p \in \mathcal{P}$.
- $\mathcal{W}$: terms having a subterm of the form $[y : W]K$ or wK, called a *bad kind*.
- $\mathcal{B}$: terms containing $\Box$.

Lemma 10 (α) No term in $\mathcal{B}$ is typable.
(β) If $\Gamma \vdash t : \Box$ then $t \in \mathcal{K}$.
(γ) If $t\theta \in \mathcal{B}$ then $t \in \mathcal{B}$ or $x\theta \in \mathcal{B}$ for some x.
(δ) If $t\theta \in \mathcal{K}$ then $t \in \mathcal{K}$ or $x\theta \in \mathcal{K}$ for some x.

Proof.
(α) $\Box$ is not typable and every subterm of a typable term is typable.
(β) By induction on the size of t (no conversion can take place since $\Box$ is not typable).
(γ) Trivial.
(δ) If $t\theta \in \mathcal{K}$ and $t \notin \mathcal{K}$ then $t = (\vec{x} : \vec{T})x$ with $x\theta \in \mathcal{K}$.

□

Lemma 11 If, for every rule $l \to r \in \mathcal{R}$, $r \notin \mathcal{B} \cup \mathcal{K} \cup \mathcal{W}$, then:
(a) If $t \to t'$ and $t' \in \mathcal{B}$ then $t \in \mathcal{B}$.
(b) If $\Box\ \mathcal{C}^*_\Gamma\ T$ then $T = \Box$.
(c) If $K \in \mathcal{K}$ and $\Gamma \vdash K : L$ then $L = \Box$.
(d) No term in $\mathcal{W}$ is typable.
(e) If $t \to K \in \mathcal{K}$ then $t \in \mathcal{K} \cup \mathcal{W}$.
(f) If $t \to t' \in \mathcal{W}$ then $t \in \mathcal{W}$.
(g) If $\Gamma \vdash T : s$ and $T \to^* K \in \mathcal{K}$ then $T \in \mathcal{K}$ and $s = \Box$.
(h) If $T\ \mathbb{C}^*_\Gamma\ K$ and $\Gamma \vdash K : \Box$ then $\Gamma \vdash T : \Box$ and $T \in \mathcal{K}$.
(i) If $(\vec{x} : \vec{T}) \star\ \mathbb{C}^*_\Gamma\ (\vec{y} : \vec{U})\star$ then $|\vec{x}| = |\vec{y}|$ and, for all i, $T_i\ \mathbb{C}^*_{\Gamma_i}\ U_i\{\vec{y} \mapsto \vec{x}\}$ with $\Gamma_i = \Gamma, x_1 : T_1, \ldots, x_i : T_i$.
(j) If $T\ \mathbb{C}^*_\Gamma\ T'$ and $\Gamma \vdash T : \star$ then $\Gamma \vdash T' : \star$.
(k) If $\Gamma \vdash t : T$ and $t \in \mathcal{P}$ then $T \in \mathcal{K}$.
(l) If $\Gamma \vdash t : K$ and $\Gamma \vdash K : \Box$ then $t \in \mathcal{P}$.

Proof.
(a) Assume that $t \to^p t'$ and $t'|_q = \Box$. If $p \sharp q$ then $t|_q = \Box$ and $t \in \mathcal{B}$. Otherwise, $p \leq q$. If $t|_p = [x : U]v\ u$ and $t'|_p = v\{x \mapsto u\}$ then, by (γ), $v \in \mathcal{B}$ or $u \in \mathcal{B}$. Thus, $t \in \mathcal{B}$. Now, if $t|_p = l\sigma$, $t'|_p = r\sigma$ and $l \to r \in \mathcal{R}$ then, by (γ), $r \in \mathcal{B}$ or $x\sigma \in \mathcal{B}$ for some x. Since $r \notin \mathcal{B}$, $x\sigma \in \mathcal{B}$ and $t \in \mathcal{B}$.
(b) Assume that $\Box \downarrow T'\ \mathbb{C}^*_\Gamma\ T$. Then, $T' \to^* \Box$ and $\Gamma \vdash T' : s$. By (a), $T' \in \mathcal{B}$ and T' cannot be typable. Thus, $T = \Box$.
(c) By induction on the size of K. If $K = \star$ then, by inversion, $\Box\ \mathcal{C}^*_\Gamma\ L$ and, by (b), $L = \Box$. If $K = (x : T)K'$ then, by inversion, $\Gamma, x : T \vdash K' : s$ and $s\ \mathcal{C}^*_\Gamma\ L$. By induction hypothesis, $s = \Box$ and, by (b), $L = \Box$.
(d) Assume that $\Gamma \vdash [y : W]K : T$. By inversion, $\Gamma, y : W \vdash K : L$ and $\Gamma \vdash (y : W)L : s$. By (c), $L = \Box$ and $(y : W)L$ cannot be typable. Assume now that $\Gamma \vdash wK : T$. By inversion, $\Gamma \vdash w : (x : L)V$, $\Gamma \vdash K : L$ and $\Gamma \vdash (x : L)V : s$. By (c), $L = \Box$ and $(x : L)V$ cannot be typable.
(e) Assume that $t \to K \in \mathcal{K}$ and $t \notin \mathcal{K}$. We prove that $t \in \mathcal{W}$ by induction on the size

of t. The only possible cases are $t = (x : T)u$, $t = [x : U]v\ u$ if $t \to_\beta K$, and $t = l\sigma$ with $l \to r \in \mathcal{R}$ if $t \to_{\mathcal{R}} K$. If $t = (x : T)u$ then $K = (x : T)L$ and $u \to L$. By induction hypothesis, $u \in \mathcal{W}$. If $t = [x : U]v\ u$ then $K = v\{x \mapsto u\}$. By (δ), either $v \in \mathcal{K}$ or $u \in \mathcal{K}$. In both cases, $t \in \mathcal{W}$. Assume now that $t = l\sigma$ with $l \to r \in \mathcal{R}$. Then, $K = r\sigma$. By (δ), either $r \in \mathcal{K}$ or $x\sigma \in \mathcal{K}$ for some x. Since $r \notin \mathcal{K}$, $x\sigma \in \mathcal{K}$ and $t = l\sigma \in \mathcal{W}$ since x is the argument of some symbol (l is algebraic).

(f) Assume that $t \to^p t' \in \mathcal{W}$, $t'|_q = wK$ and $K \in \mathcal{K}$ (the case $t'|_q = [x : w]K$ is dealt with in the same way). There are several cases:
- $q \sharp p$. Then, $t|_q = wK$ and $t \in \mathcal{W}$.
- $q < p$.
  - $p = q1m$. Then, $t|_q = w'K$ with $w' \to w$ and $t \in \mathcal{W}$.
  - $p = q2m$. Then, $t|_q = wu$ with $u \to K \in \mathcal{K}$. By (e), $u \in \mathcal{K} \cup \mathcal{W}$. Thus, $t \in \mathcal{W}$.
- $q \geq p$. Then, $q = pm$. Assume that $t|_p = l\sigma$, $t'|_p = r\sigma$ and $l \to r \in \mathcal{R}$ (the case $t \to_\beta t'$ is dealt with in the same way). Let $\{p_1, \ldots, p_n\} = \{p \in \mathrm{Pos}(x, r) \mid x \in \mathrm{FV}(r)\}$. There are saveral cases:
  - $m \sharp p_i$ for all i, or $m < p_i$ for some i. Then, $r|_m\sigma = wK$, $r = uv$ and $v\sigma = K$. By (δ), $v \in \mathcal{K}$ or $x\sigma \in \mathcal{K}$ for some x. If $v \in \mathcal{K}$ then $r \in \mathcal{W}$, which is not possible. Thus, $x\sigma \in \mathcal{K}$ and $l\sigma \in \mathcal{W}$.
  - $m \geq p_i$ for some i. Then, there is $x \in \mathrm{FV}(l)$ such that $x\sigma \in \mathcal{W}$. Thus, $t \in \mathcal{W}$.

(g) By (e) and (f), if $T \to^* K \in \mathcal{K}$ then $T \in \mathcal{K} \cup \mathcal{W}$. Since $\Gamma \vdash T : s$, $T \notin \mathcal{W}$. Thus, $T \in \mathcal{K}$ and $s = \Box$.

(h) By induction on the number of conversions between T and K. Assume that $\Gamma \vdash T : s$, $T \downarrow K$ and $\Gamma \vdash K : \Box$. Then, there is $K' \in \mathcal{K}$ such that $K \to^* K'$ and $T \to^* K'$. By (g), $T \in \mathcal{K}$ and $s = \Box$.

(i) By (h), all the intermediate well-typed terms between $K = (\vec{x} : \vec{T})\star$ and $L = (\vec{y} : \vec{U})\star$ are kinds and, if $K \downarrow L$ then, clearly, $|\vec{x}| = |\vec{y}|$ and $T_i \downarrow U_i\{\vec{x} \mapsto \vec{y}\}$ for all i.

(j) Immediate consequence of (i).

(k) By induction on $\Gamma \vdash t : T$.

(l) By induction on $\Gamma \vdash t : K$.

□

Lemma 12 Given a rule $l \to r$ with $l = f\vec{l}$, $f : (\vec{x} : \vec{T})U$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$, $r \notin \mathcal{B} \cup \mathcal{K} \cup \mathcal{W}$ if there is an environment Γ and a substitution ρ such that $\Gamma \vdash l\rho : U\gamma\rho$ and $\Gamma \vdash r : U\gamma\rho$.

Proof. Since r is typable, $r \notin \mathcal{B} \cup \mathcal{W}$. We now prove that $r \notin \mathcal{K}$. Since $\Gamma \vdash l\rho : U\gamma\rho$, by inversion, we get that $\gamma\rho : \Gamma_f \rightsquigarrow \Gamma$. Since $\vdash \tau_f : s_f$, by inversion, we get that $\Gamma_f \vdash U : s_f$. So, by substitution, $\Gamma \vdash U\gamma\rho : s_f$. Now, if $r \in \mathcal{K}$ then, by (c), $U\gamma\rho = \Box$ but $\Box$ is not typable. Therefore, $r \notin \mathcal{K}$. □

Theorem 13 (Subject reduction for h) (Barbanera *et al.* 1997) Assume that no right hand-side is in $\mathcal{B} \cup \mathcal{K} \cup \mathcal{W}$. Then, the restriction $\beta^{P\omega}$ of β to the redexes $[x : T]U\ t \in \mathcal{P}$ preserves typing. Therefore, h preserves typing on terms of type $\star$.

Proof. The proof is as usual by induction on $\Gamma \vdash t : T$ and by proving at the same time that, if $\Gamma \to_{\beta^{P\omega}} \Gamma'$, then $\Gamma' \vdash t : T$. The only difficult case is the case of a head-reduction $[x : U']v\ u \to_{\beta^{P\omega}} v\{x \mapsto u\}$ with $\Gamma \vdash [x : U']v : (x : U)V$ and $\Gamma \vdash u : U$. We must prove that $\Gamma \vdash v\{x \mapsto u\} : V\{x \mapsto u\}$. By inversion, we have $\Gamma, x : U' \vdash v : V'$ with $(x : U')V'\ \mathbb{C}^*_\Gamma\ (x : U)V$. Since $v \in \mathcal{P}$, by (k), $V' \in \mathcal{K}$. Therefore, by Lemma 11 (h) and (i), $(x : U)V \in \mathcal{K}$, $U'\ \mathbb{C}^*_\Gamma\ U$ and $V'\ \mathbb{C}^*_{\Gamma,x:U}\ V$. Hence, by environment conversion and type conversion, $\Gamma, x : U \vdash v : V$ and, by substitution, $\Gamma \vdash v\{x \mapsto u\} : V\{x \mapsto u\}$.

Now, if $\Gamma \vdash t : \star$ then, by (l), $t = [x : U]vu\vec{t} \in \mathcal{P}$ and $v \in \mathcal{P}$. So, if $t \to_h t'$ then $t \to_{\beta^{P\omega}} t'$ and $\Gamma \vdash t' : \star$. □

Lemma 14 (Commutation) If $t \to_h^* u$ and $t \to_{\mathcal{R}}^* v$ then there exists w such that $u \to_{\mathcal{R}}^* w$ and $v \to_h^* w$.

Proof. By induction on the number of h-steps, it suffices to prove that, if $[x : U]v\ u \to_h v\{x \mapsto u\}$ and $[x : U]v\ u \to_{\mathcal{R}}^* t$, then there exists w such that $v\{x \mapsto u\} \to_{\mathcal{R}}^* w$ and $t \to_h w$. Since left hand-sides of rules are algebraic, t is of the form $[x : U']v'\ u'$ with $U \to_{\mathcal{R}}^* U'$, $v \to_{\mathcal{R}}^* v'$ and $u \to_{\mathcal{R}}^* u'$. So, it suffices to take $w = v'\{x \mapsto u'\}$. □

Lemma 15 (Postponement) Assume that no right hand-side is in $\mathcal{B} \cup \mathcal{K} \cup \mathcal{W}$ and that the right hand-side of every type-level rule is either a product or a predicate symbol application. If $\Gamma \vdash t : \star$ and $t \to_{\mathcal{R}}^* u \to_h^* v$ then there exists w such that $t \to_h^* w \to_{\mathcal{R}}^* v$.

Proof. By induction on the number of h-steps. Assume that $t \to_{\mathcal{R}}^* u \to_h^* u' \to_h v$. By induction hypothesis, there exists w' such that $t \to_h^* w' \to_{\mathcal{R}}^* u'$. By subject reduction on types, $\Gamma \vdash w' : \star$. So, by (l), w' is either of the form $(x : U)V$, $x\vec{t}$, $f\vec{t}$ with $f \in \mathcal{F}^\square$, or $[x : B]ab\vec{t}$. Since $w' \to_{\mathcal{R}}^* u' \to_h v$, w' cannot be of the form $(x : U)V$ or $x\vec{t}$. Since right hand-sides of type-level rules are either a product or a predicate symbol application, w' cannot be of the form $f\vec{t}$. Therefore, $w' = [x : B]ab\vec{t}$, $u' = [x : B']a'b'\vec{t'}$ with $B, a, b, \vec{t} \to_{\mathcal{R}}^* B', a', b', \vec{t'}$, and $v = a'\{x \mapsto b'\}\vec{t'}$. Hence, by taking $w = a\{x \mapsto b\}\vec{t}$, we have $t \to_h w' \to_h w \to_{\mathcal{R}}^* v$. □

Theorem 16 (Subject reduction for β) If no right hand-side is in $\mathcal{B} \cup \mathcal{K} \cup \mathcal{W}$ and the right hand-side of every type-level rule is a symbol application then β preserves typing.

Proof. The proof is as usual by induction on $\Gamma \vdash t : T$ and by proving that, if $\Gamma \to_\beta \Gamma'$, then $\Gamma' \vdash t : T$. The only difficult case is the case of a head-reduction $[x : U']v\ u \to_\beta v\{x \mapsto u\}$ with $\Gamma \vdash [x : U']v : (x : U)V$ and $\Gamma \vdash u : U$. We must prove that $\Gamma \vdash v\{x \mapsto u\} : V\{x \mapsto u\}$. We already know that it is true when v is a predicate. We must now prove it when v is an object, that is, when $\Gamma \vdash (x : U)V : \star$. By inversion, we have $\Gamma \vdash [x : U']v : (x : U')V'$ with $(x : U')V'\ \mathbb{C}^*_\Gamma\ (x : U)V$. By Lemma 11 (j), we have all the intermediate well-typed terms between $(x : U')V'$ and $(x : U)V$ of type $\star$. Without loss of generality, we can assume that $T_0 = (x : U')V' \downarrow_\beta T_1 \downarrow_{\mathcal{R}} T_2 \downarrow_\beta \ldots T_n = (x : U)V$. Let T_i' be the common reduct between T_i and T_{i+1}. We now prove by induction on the number of conversions that there is a sequence of well-typed product terms $\pi_1, \ldots, \pi_n$ such that $\pi_0 = T_0 \downarrow_\beta \pi_1 \downarrow_{\mathcal{R}} \pi_2 \downarrow_\beta \ldots \pi_n = T_n$.

Since T_0 is a product, $\pi_0' = T_0'$ is also a product. Since $T_1 \to_\beta^* \pi_0'$, by standardization, there is a product term π_1 such that $T_1 \to_h^* \pi_1 \to_{\not h}^* \pi_0'$. Since h has the subject reduction property on types, π_1 is well-typed. Now, since $T_1 \to_{\mathcal{R}}^* T_1'$, by commutation, there is a product term π_1' such that $\pi_1 \to_{\mathcal{R}}^* \pi_1'$ and $T_1' \to_h^* \pi_1'$. Furthermore, since $T_2 \to_{\mathcal{R}}^* T_1'$, by postponement, there is a term t such that $T_2 \to_h^* t \to_{\mathcal{R}}^* \pi_1'$. Since h has the subject reduction property on types, t is a well-typed term of type $\star$. We now proceed by case on t.

- If t is an abstraction $[x : T]w$ then, by inversion, there is W such that $(y : T)W \; \mathbb{C}_\Gamma^* \star$. By Lemma 11 (h) and (i), this is not possible.
- If t is an application but not a symbol application then, since left hand-sides of rules are algebraic, π_1' is an application, which is not possible either.
- If t is a symbol application then, since right hand-sides of type-level rules are symbol applications, π_1' is a symbol application too, which is not possible either.
- Therefore, t is a well-typed product term π_2.

Now, since $T_2 \to_\beta^* T_2'$ and β is confluent, there is a product term π_2' such that $\pi_2 \to_{\not h}^* \pi_2'$ and $T_2' \to_\beta^* \pi_2'$, and we can now conclude by induction. □

4. Logical consistency

In the case of the pure Calculus of Constructions without symbols and rewrite rules, logical consistency easily follows from normalization by proving that there can be no normal proof of $\bot = (\alpha : \star)\alpha$ in the empty environment (Barendregt 1992). But, having symbols and rewrite rules is like having hypothesis and axioms. Thus, in this case, logical consistency does not directly follow from normalization. We can however give general conditions ensuring logical consistency:

Theorem 17 (Logical consistency) Assume that $\to$ is confluent and that every object symbol f satisfies one of the following conditions:

(1) $f : (\vec{x} : \vec{T})C\vec{v}$ with $C \in \mathcal{CF}^\square$,
(2) $f : (\vec{x} : \vec{T})T_i$,
(3) $f : (x_1 : T_1)\dots(x_n : T_n)U$ with $x_n \notin \mathrm{FV}^\square(U)$ and, for all normal substitution $\gamma : (\vec{x} : \vec{T}) \rightsquigarrow (\alpha : \star)$, $f\vec{x}\gamma$ is reducible.

Then, there is no normal proof of $\bot = (\alpha : \star)\alpha$ in the empty environment. Therefore, if $\to$ is also normalizing, then there is no proof of $\bot$ in the empty environment.

Proof. Assume that $\vdash t : \bot$, t is normal and of minimal size, that is, there is no term u smaller than t such that $\vdash u : \bot$. For typing reasons, t cannot be a sort or a product. Assume that t is an application. Since t is typable in the empty environment, it cannot have free variables and, since t is normal, it must be of the form $f\vec{t}$. Assume that $|\vec{t}| = k$ and that f is of type $(\vec{x} : \vec{T})U$ with $|\vec{x}| = n$. Let $\gamma_i = \{x_1 \mapsto t_1, \dots, x_i \mapsto t_i\}$ $(i \leq n)$.

(1) In this case, $k \leq n$ since f cannot be applied to more than n arguments. Indeed, if f is applied to $n+1$ arguments then, by inversion, $\vdash ft_1 \dots t_n : (x_{n+1} : T_{n+1})V$. But, since $\vdash ft_1 \dots t_n : C\vec{v}\gamma_n$, by convertibility of types and confluence, we must

have $(x_{n+1} : T_{n+1})V \downarrow C\vec{v}\gamma_n$, which is not possible. Thus, $k \leq n$ and $(x_{k+1} : T_{k+1}\gamma_k)\dots(x_n : T_n\gamma_k)C\vec{v}\gamma_k \downarrow \bot$, which is not possible either.

(2) There are 2 cases:
- $k < n$. Since $\vdash f\vec{t} : (x_{k+1} : T_{k+1}\gamma_k)\dots(x_n : T_n\gamma_k)T_i\gamma_k$, we must have $n = k+1$ and, by taking $x_n = \alpha$, $T_n\gamma_k \downarrow \star$ and $T_i\gamma_k \downarrow \alpha$. Hence $T_i\gamma_k \to^* \alpha$ but $T_i\gamma_k$ is closed since $\mathrm{FV}(T_i) \subseteq \{x_1, \dots, x_{i-1}\}$, γ_k is closed and $i - 1 \leq k$. So, $T_i\gamma_k \to^* \alpha$ is not possible.
- $k \geq n$. We have $\vec{t} = \vec{u}\vec{v}$ with $|\vec{u}| = n$. Let $p = k-n$. By inversion, there is a sequence of products $(y_1 : V_1)W_1$, …, $(y_p : V_p)W_p$ such that $T_i\gamma_n = U\gamma_n \downarrow (y_1 : V_1)W_1$, for all $i < p$, $W_i\{y_i \mapsto v_i\} \downarrow (y_{i+1} : V_{i+1})W_{i+1}$, and $W_p\{y_p \mapsto v_p\} \downarrow \bot$. Then, $\vdash u_i\vec{v} : \bot$ and $u_i\vec{v}$ is smaller than t.

(3) If $k \geq n$ then t is reducible, which is not possible. If $k < n$ then $n = k+1$, $x_n = \alpha$ and $U\gamma_k \to^* \alpha$. But $\mathrm{FV}(U) \subseteq \{x_1, \dots, x_k, \alpha\}$ and γ_k is closed. So, $x_n \in \mathrm{FV}^{\Box}(U)$, which is excluded.

Assume now that $t = [\alpha : T]v$. Then, by inversion, we must have $\alpha : T \vdash v : V$ and $(\alpha : T)V \downarrow (\alpha : \star)\alpha$. Therefore, $T = \star$, $V = \alpha$ and $\alpha : \star \vdash v : \alpha$. For typing reasons, v cannot be a sort, a product or an abstraction. Since it is normal, it must be of the form $x\vec{u}$ with x a variable, or of the form $f\vec{t}$. Since α is the only variable that may freely occur in v, $x = \alpha$. Since α can be applied to no argument, $v = \alpha$. Then, we get $\alpha : \star \vdash \alpha : \alpha$, which is not possible. Therefore, v is of the form $f\vec{t}$.

(1) In this case, $k \leq n$ since f cannot be applied to more than n arguments. Thus, $(x_{k+1} : T_{k+1}\gamma_k)\dots(x_n : T_n\gamma_k)C\vec{v}\gamma_k \downarrow \alpha$, which is not possible.

(2) If $k < n$ then $(x_{k+1} : T_{k+1}\gamma_k)\dots(x_n : T_n\gamma_k)T_i\gamma_k \downarrow \alpha$, which is not possible. Thus, $\vec{t} = \vec{u}\vec{v}$ with $|\vec{u}| = n$. Let $p = k - n$. By inversion, there is a sequence of products $(y_1 : V_1)W_1$, …, $(y_p : V_p)W_p$ such that $T_i\gamma_n = U\gamma_n \downarrow (y_1 : V_1)W_1$, for all $i < p$, $W_i\{y_i \mapsto v_i\} \downarrow (y_{i+1} : V_{i+1})W_{i+1}$, and $W_p\{y_p \mapsto v_p\} \downarrow \alpha$. Then, $\vdash [\alpha : \star]u_i\vec{v} : \bot$ and $[\alpha : \star]u_i\vec{v}$ is smaller than t.

(3) In this case too, $k \geq n$. Thus, t is reducible, which is not possible.

□

Note that, as opposed to the third condition, the first two conditions do not care about the rewrite rules defining f.

To see the interest of the third condition, consider the following example. Assume that the only symbols of the calculus are $nat : \star$, $0 : nat$, $s : nat \to nat$ and $rec : (P : nat \to \star)$ $P0 \to ((n : nat)Pn \to P(sn)) \to (n : nat)Pn$ defined by the usual rules for recursors:

$$\begin{array}{rcl} rec\ P\ u\ v\ 0 & \to & u \\ rec\ P\ u\ v\ (s\ n) & \to & v\ n\ (rec\ P\ u\ v\ n) \end{array}$$

This calculus is confluent since the combination of an orthogonal system (the recursor rules) with the β-reduction preserves confluence. In this calculus, it is possible to express any function whose existence is provable in intuitionistic higher-order arithmetic.

Now, let us look at the normal terms of type nat in the environment $\alpha : \star$. Let $\mathcal{N}$ be the set of these terms. A term in $\mathcal{N}$ cannot be a sort, a product, an abstraction, nor a variable. It can only be of the form 0, $(s\ t)$ with t itself in $\mathcal{N}$, or of the form $(rec\ P\ u\ v\ t\ \vec{u})$

with $t \in \mathcal{N}$ also. But the last case is not possible since, at some point, the argument t of $(rec\ P\ u\ v\ t)$ must be of the form 0 or $(s\ t')$, and hence $(rec\ P\ u\ v\ t)$ must be reducible. Therefore, all the normal terms of type *nat* typable in $\alpha : \star$ must be of the form 0 or $(s\ t)$, and if t is such a term then $(rec\ P\ u\ v\ t)$ is reducible. We also say that functions defined by induction are *completely defined* (Guttag and Horning 1978; Thiel 1984; Kounalis 1985; Coquand 1992). Therefore, after the previous theorem, this calculus is consistent.

This may certainly be extended to the Calculus of Inductive Constructions and even to the Calculus of Inductive Constructions extended with functions defined by rewrite rules whenever all the symbols are completely defined.

5. Conditions of Strong Normalization

We now present the conditions of strong normalization.

5.1. *Inductive types and constructors*

Until now we made few hypothesis on symbols and rewrite rules. However, Mendler (Mendler 1987) showed that the extension of the simply-typed λ-calculus with recursion on inductive types is strongly normalizing if and only if the inductive types satisfy some positivity condition.

A base type T occurs positively in a type U if all the occurrences of T in U are on the left of an even number of $\Rightarrow$. A type T is positive if T occurs positively in the type of the arguments of its constructors. Usual inductive types like natural numbers and lists of natural numbers are positive.

Now, let us see an example of a non-positive type T. Let U be a base type. Assume that T has a constructor c of type $(T \Rightarrow U) \Rightarrow T$. T is not positive because T occurs at a negative position in $T \Rightarrow U$. Consider now the function p of type $T \Rightarrow (T \Rightarrow U)$ defined by the rule $p(cx) \to x$. Let $\omega = \lambda x.(px)x$ of type $T \Rightarrow U$. Then the term $\omega(c\omega)$ of type U is not normalizable:

$$\omega(c\omega)\ \to_\beta\ p(c\omega)(c\omega)\ \to_{\mathcal{R}}\ \omega(c\omega)\ \to_\beta\ \ldots$$

In the case where $U = \star$, we can interpret this as Cantor's theorem: there is no surjection from a set T to the set of its subsets $T \Rightarrow \star$. In this interpretation, p is the natural injection between T and $T \Rightarrow \star$. Saying that p is surjective is equivalent to saying (with the Axiom of Choice) that there exists c such that $p \circ c$ is the identity, that is, such that $p(cx) \to x$. In (Dowek 1999), Dowek shows that such an hypothesis is incoherent. Here, we show that this is related to the non-normalization of non-positive inductive types.

Mendler also gives a condition, strong positivity, in the case of dependent and polymorphic types. A similar but more restrictive notion, called strict positivity, is used by Coquand and Paulin in the Calculus of Inductive Constructions (Coquand and Paulin-Mohring 1988).

Hereafter we introduce the more general notion of *admissible inductive structure*. In

particular, we do not consider that a constructor must be constant: it is possible to have rewrite rules on constructors. This allows us to formalize quotient types like the type int of integers by taking $0 : int$ for zero, $s : int \Rightarrow int$ for successor, and $p : int \Rightarrow int$ for predecessor, together with the rules:

$$\begin{array}{rcl} s\ (p\ x) & \to & x \\ p\ (s\ x) & \to & x \end{array}$$

Definition 18 (Inductive structure) An *inductive structure* is given by:

- a precedence $\geq_{\mathcal{C}}$ on $\mathcal{CF}^{\Box}$,
- for every $C : (\vec{x} : \vec{T})\star$ in $\mathcal{CF}^{\Box}$, a set $\mathrm{Mon}(C) \subseteq \{i \leq |\vec{x}| \mid x_i \in \mathcal{X}^{\Box}\}$ for the *monotonic arguments* of C,
- for every $f : (\vec{y} : \vec{U})C\vec{v}$ with $C \in \mathcal{CF}^{\Box}$, a set $\mathrm{Acc}(f) \subseteq \{1, \ldots, |\vec{y}|\}$ for the *accessible positions* of f.

For convenience, we assume that $\mathrm{Mon}(f) = \emptyset$ if $f \notin \mathcal{CF}^{\Box}$, and $\mathrm{Acc}(f) = \emptyset$ if f is not of type $(\vec{y} : \vec{U})C\vec{v}$ with $C \in \mathcal{CF}^{\Box}$.

The accessible positions of f denote the arguments of f that one can use in the right hand-sides of rules. The monotonic arguments of C denote the parameters in which C is monotonic.

Definition 19 (Positive and negative positions) The set of *positive positions* in t, $\mathrm{Pos}^+(t)$, and the set of *negative positions* in t, $\mathrm{Pos}^-(t)$, are simultaneously defined by induction on the structure of t:

- $\mathrm{Pos}^{\delta}(s) = \mathrm{Pos}^{\delta}(x) = \{\varepsilon \mid \delta = +\}$,
- $\mathrm{Pos}^{\delta}((x : U)V) = 1.\mathrm{Pos}^{-\delta}(U) \cup 2.\mathrm{Pos}^{\delta}(V)$,
- $\mathrm{Pos}^{\delta}([x : U]v) = 2.\mathrm{Pos}^{\delta}(v)$,
- $\mathrm{Pos}^{\delta}(tu) = 1.\mathrm{Pos}^{\delta}(t)$ if $t \neq f\vec{t}$,
- $\mathrm{Pos}^{\delta}(f\vec{t}) = \{1^{|\vec{t}|} \mid \delta = +\} \cup \bigcup\{1^{|\vec{t}|-i}2.\mathrm{Pos}^{\delta}(t_i) \mid i \in \mathrm{Mon}(f)\}$,

where $\delta \in \{-, +\}$, $-+ = -$ and $-- = +$ (usual rule of signs).

Definition 20 (Admissible inductive structures) An inductive structure is *admissible* if, for all $C \in \mathcal{CF}^{\Box}$, for all $f : (\vec{y} : \vec{U})C\vec{v}$, and for all $j \in \mathrm{Acc}(f)$:[††]

(I3) $\forall D \in \mathcal{CF}^{\Box}, D =_{\mathcal{C}} C \Rightarrow \mathrm{Pos}(D, U_j) \subseteq \mathrm{Pos}^+(U_j)$
(symbols equivalent to C must be at positive positions),

(I4) $\forall D \in \mathcal{CF}^{\Box}, D >_{\mathcal{C}} C \Rightarrow \mathrm{Pos}(D, U_j) = \emptyset$
(no symbol greater than C can occur in U_j),

(I5) $\forall F \in \mathcal{DF}^{\Box}, \mathrm{Pos}(F, U_j) = \emptyset$
(no defined symbol can occur in U_j),

[††] In (Blanqui 2001), we give 6 conditions, (I1) to (I6), for defining what is an admissible inductive structure. But we found that (I1) can be eliminated if we modify (I2) a little bit. This is why, in the following definition, there is no (I1) and (I2) is placed after (I6).

(I6) $\forall Y \in \mathrm{FV}^{\Box}(U_j), \exists \iota_Y, v_{\iota_Y} = Y$
(predicate variables must be parameters of C),
(I2) $\forall Y \in \mathrm{FV}^{\Box}(U_j), \iota_Y \in \mathrm{Mon}(C) \Rightarrow \mathrm{Pos}(Y, U_j) \subseteq \mathrm{Pos}^{+}(U_j)$
(monotonic arguments must be at positive positions).

For instance, with $list : \star \Rightarrow \star$, $nil : (A : \star)listA$ and $cons : (A : \star)A \Rightarrow listA \Rightarrow listA$, $\mathrm{Mon}(list) = \{1\}$, $\mathrm{Acc}(nil) = \{1\}$ and $\mathrm{Acc}(cons) = \{1, 2, 3\}$ is an admissible inductive structure. If we add $tree : \star$ and $node : list\ tree \Rightarrow tree$ with $\mathrm{Mon}(list) = \{1\}$, $\mathrm{Mon}(tree) = \emptyset$ and $\mathrm{Acc}(node) = \{1\}$, we still have an admissible structure.

The condition (I6) means that the predicate arguments of a constructor must be parameters of their type. A similar condition appears in the works of Stefanova (Stefanova 1998) ("safeness") and Walukiewicz (Walukiewicz-Chrząszcz 2002) ("$\star$-dependency"). On the other hand, in the Calculus of Inductive Constructions (CIC) (Werner 1994), there is no such restriction.

We distinguish several kinds of inductive types.

Definition 21 (Primitive, basic and strictly-positive predicates)
A constant predicate symbol C is:

- *primitive* if for all $D =_{\mathcal{C}} C$, for all $f : (\vec{y} : \vec{U})D\vec{w}$ and for all $j \in \mathrm{Acc}(f)$, $U_j = E\vec{t}$ with $E <_{\mathcal{C}} D$ and E primitive, or $U_j = E\vec{t}$ with $E =_{\mathcal{C}} D$.
- *basic* if for all $D =_{\mathcal{C}} C$, for all $f : (\vec{y} : \vec{U})D\vec{w}$ and for all $j \in \mathrm{Acc}(f)$, if $E =_{\mathcal{C}} D$ occurs in U_j then U_j is of the form $E\vec{t}$.
- *strictly positive* if for all $D =_{\mathcal{C}} C$, for all $f : (\vec{y} : \vec{U})D\vec{w}$ and for all $j \in \mathrm{Acc}(f)$, if $E =_{\mathcal{C}} D$ occurs in U_j then $U_j = (\vec{z} : \vec{V})E\vec{t}$ and no $D' =_{\mathcal{C}} D$ occurs in $\vec{V}$.

Primitive predicates are basic and basic predicates are strictly positive. Note that primitive predicates not only include the usual first-order data types. They also include some dependent type like the type of lists of fixed length. On the other hand, the type of polymorphic lists is basic but not primitive.

The strictly positive predicates are the predicates allowed in the Calculus of Inductive Constructions (CIC). For example, this includes the type *ord* of Brouwer's ordinals whose constructors are $0 : ord$, $s : ord \Rightarrow ord$ and $lim : (nat \Rightarrow ord) \Rightarrow ord$, the process algebra μCRL which can be formalized as a type *proc* with a choice operator $\Sigma : (data \Rightarrow proc) \Rightarrow proc$ (Sellink 1993), or the type *form* of the formulas of first-order predicate calculus whose constructors are $\neg : form \Rightarrow form$, $\vee : form \Rightarrow form \Rightarrow form$ and $\forall : (term \Rightarrow form) \Rightarrow form$.

For the moment, we do not forbid non-strictly positive predicates but the conditions we describe in the next section do not allow the definition of functions by recursion on such predicates. Yet, these predicates can be very useful as shown in (Matthes 2000) or in (Abel 2001). In (Matthes 2000), a type *cont* with the constructors $D : cont$ and $C : ((cont \Rightarrow list) \Rightarrow list) \Rightarrow cont$, representing continuations, is used to define a breadth-first label listing function on labelled binary trees. In particular, it uses a function $ex : cont \Rightarrow list$ defined by the rules:

$$\begin{array}{rcl} ex\ D & \to & nil \\ ex\ (C\ f) & \to & f\ ex \end{array}$$

It is not clear how to define a syntactic condition ensuring the strong normalization of such a definition: in the right hand-side of the second rule, ex is explicitly applied to no argument smaller than f. And, although ex can only be applied to subterms of reducts of f, not every subterm of a "computable" term (notion used for proving strong normalization) is *a priori* computable (see Section 5.2.2).

5.2. *General Schema*

5.2.1. *Higher-order rewriting* Which conditions on rewrite rules would ensure the strong normalization of $\to\!=\!\to_{\mathcal{R}} \cup \to_\beta$? Since the works of Breazu-Tannen and Gallier (Breazu-Tannen and Gallier 1989) and Okada (Okada 1989) on the simply-typed λ-calculus or the polymorphic λ-calculus, and later the works of Barbanera (Barbanera 1990) on the Calculus of Constructions and of Dougherty (Dougherty 1991) on the untyped λ-calculus, it is well known that adding first-order rewriting to typed λ-calculi preserves strong normalization. This comes from the fact that first-order rewriting cannot create β-redexes. We will prove that this result can be extended to predicate-level rewriting if some conditions are fulfilled.

However, there are many useful functions whose definition do not enter the first-order framework, either because some arguments are not primitive (the concatenation function *app* on polymorphic lists), or because their definition uses higher-order features like the function $map : (A : \star)(B : \star)(A \Rightarrow B) \Rightarrow listA \Rightarrow listB$ which applies a function to every element of a list:

$$\begin{array}{rcl} map\ A\ B\ f\ (nil\ A') & \to & nil\ B \\ map\ A\ B\ f\ (cons\ A'\ x\ \ell) & \to & cons\ B\ (f\ x)\ (map\ A\ B\ f\ \ell) \\ map\ A\ B\ f\ (app\ A'\ \ell\ \ell') & \to & app\ B\ (map\ A\ B\ f\ \ell)\ (map\ A\ B\ f\ \ell') \end{array}$$

This is also the case of recursors like the recursor on natural numbers $natrec : (A : \star)$ $A \Rightarrow (nat \Rightarrow A \Rightarrow A) \Rightarrow nat \Rightarrow A$:

$$\begin{array}{rcl} natrec\ A\ x\ f\ 0 & \to & x \\ natrec\ A\ x\ f\ (s\ n) & \to & f\ n\ (natrec\ A\ x\ f\ n) \end{array}$$

and of induction principles (recursors are just non-dependent versions of the corresponding induction principles), like the induction principle on natural numbers $natind : (P :$ $nat \Rightarrow \star)P0 \Rightarrow ((n : nat)Pn \Rightarrow P(sn)) \Rightarrow (n : nat)Pn$:

$$\begin{array}{rcl} natind\ P\ h_0\ h_s\ 0 & \to & h_0 \\ natrec\ P\ h_0\ h_s\ (s\ n) & \to & h_s\ n\ (natind\ P\ h_0\ h_s\ n) \end{array}$$

The methods used by Breazu-Tannen and Gallier (Breazu-Tannen and Gallier 1989) or Dougherty (Dougherty 1991) cannot be applied to our calculus since, on the one hand, higher-order rewriting can create β-redexes and, on the other hand, rewriting is included in the type conversion rule (conv), hence more terms are typable. But there exists other methods, available in the simply-typed λ-calculus only or in richer type systems, for proving the termination of this kind of definitions:

- The *General Schema*, initially introduced by Jouannaud and Okada (Jouannaud and Okada 1991) for the polymorphic λ-calculus and extended to the Calculus of Constructions by Barbanera, Fernández and Geuvers (Barbanera *et al.* 1994), is an extension of the primitive recursion schema: in the right hand-side of a rule $f\vec{l} \to r$, the recursive calls to f must be done on strict subterms of $\vec{l}$. It can treat object-level rewriting with simply-typed symbols defined on primitive types. It has been reformulated and extended to strictly-positive types by Jouannaud, Okada and the author for the simply-typed λ-calculus (Blanqui *et al.* 2002) and the Calculus of Constructions (Blanqui *et al.* 1999).
- The *Higher-Order Recursive Path Ordering* (HORPO) (Jouannaud and Rubio 1999) is an extension of RPO (Plaisted 1978; Dershowitz 1982) to the simply-typed λ-calculus. It has been recently extended by Walukiewicz (Walukiewicz 2000) to the Calculus of Constructions with strictly positive types (Walukiewicz-Chrząszcz 2002). It can treat object-level rewriting with polymorphic and dependent symbols defined on strictly positive types. The General Schema can be seen as a non-recursive version of HORPO.
- It is also possible to look for an interpretation of the symbols such that the interpretation of a term strictly decreases when a rule is applied. This method, introduced by Van de Pol for the simply-typed λ-calculus (Van de Pol 1996), extends to the higher-order framework the method of interpretations known for the first-order framework (Zantema 1994). This is a very powerful method but difficult to use in practice because the interpretations are themselves higher-order and also because it is not modular: adding new rules or new symbols may require finding new interpretations.

For dealing with higher-order rewriting at the predicate-level together with polymorphic and dependent symbols and strictly-positive predicates, we have chosen to extend the method of the General Schema. For first-order symbols, we use other conditions like in (Jouannaud and Okada 1997).

5.2.2. *Definition of the schema* This method is based on Tait and Girard's method of reducibility candidates (Tait 1967; Girard *et al.* 1988) for proving the strong normalization of simply-typed or polymorphic λ-calculi. This method consists of interpretating each type as a subset of the strongly normalizable terms, the *computable* terms, and proving that each well-typed term is computable. Indeed, a direct proof of strong normalization by induction on the structure of terms does not go through because of the application case: if u and v are strongly normalizable then it is not clear how to prove that uv also is strongly normalizable.

The idea of the General Schema is then, from a left hand-side $f\vec{l}$ of rule, to define a set of terms, called the *computability closure* of $f\vec{l}$, whose elements are computable whenever the l_i's so are. Then, to prove the strong normalization, it suffices to check that, for each rule, the right hand-side belongs to the computability closure of the left hand-side.

To build the computability closure, we first define a subset of the subterms of $\vec{l}$, called the *accessible* subterms of $\vec{l}$, that are computable whenever the l_i's so are (not all the subterms of a computable term are *a priori* computable). Then, we build the computability

closure by closing the set of accessible variables of the left hand-side with computability-preserving operations.

In order to have interesting functions, we must be able to accept recursive calls and, to preserve strong normalization, recursive calls must decrease in a well-founded ordering. The strict subterm relation $\rhd$ (in fact, restricted to accessible subterms for preserving computability) is sufficient for dealing with definition on basic predicates. In the definition of *map* for instance, ℓ and ℓ' are accessible subterms of *app* $A'\ \ell\ \ell'$. But, for non-basic predicates, it is not sufficient as examplified by the following addition on Brouwer's ordinals:

$$\begin{array}{rcl} x+0 & \to & x \\ x+(s\ y) & \to & s\ (x+y) \\ x+(lim\ f) & \to & lim\ ([n:nat]x+fn) \end{array}$$

Another example is given by the following simplification rule in μCRL (Sellink 1993):

$$(\Sigma\ f)\cdot p \quad \to \quad \Sigma\ ([d:data]fd\cdot p)$$

This is why, in our conditions, we use two distinct orderings. The first one, $>_1$, is used for the arguments of basic type and the second one, $>_2$, is used for the arguments of strictly-positive type.

Finally, to have a finer control of the comparison of the arguments, to each symbol, we associate a *status* describing how to compare the arguments by using a simple combination of lexicographic and multiset comparisons (Jouannaud and Okada 1997).

Definition 22 (Accessibility) We say that $u:U$ is *accessible modulo* ρ in $t:T$, written $t:T\ \rhd_\rho\ u:U$, if $t=f\vec{u}$, $f:(\vec{y}:\vec{U})C\vec{v}$, $C\in\mathcal{CF}^\Box$, $u=u_j$, $j\in\mathrm{Acc}(f)$, $T\rho=C\vec{v}\gamma\rho$, $U\rho=U_j\gamma\rho$, $\gamma=\{\vec{y}\mapsto\vec{u}\}$ and no $D=_\mathcal{C}C$ occurs in $\vec{u}\rho$.

For technical reasons, we take into account not only the terms themselves but also their types. This comes from the fact that we are able to prove that two convertible types have the same interpretation only if the two types are computable. This may imply some restrictions on the types of the symbols.

Indeed, accessibility requires the equality (modulo the application of ρ) between canonical types and derived types (see Definition 6). More precisely, for having $t:T\rhd_\rho u:U$, T must be equal (modulo ρ) to the canonical type of t and U must be equal (modulo ρ) to the type of u derived from t. In addition, if $u:U\rhd_\rho v:V$, then U must also be equal (modulo ρ) to the canonical type of u.

Definition 23 Let $(x_i)_{i\geq 1}$ be an indexed family of variables.

Status. A *status* is a term of the form $(lex\ m_1\dots m_k)$ with $k\geq 1$ and each m_i of the form $(mul\ x_{k_1}\dots x_{k_p})$ with $p\geq 1$. The *arity* of a status *stat* is the greatest index i such that x_i occurs in *stat*.

Status assignment. A *status assignment* is an application *stat* which associates a status $stat_f$ to every $f\in\mathcal{F}$.

Predicate arguments. Let $C:(\vec{z}:\vec{V})\star$ and $\vec{u}$ with $|\vec{u}|=|\vec{z}|$. By $\vec{u}|_C$, we denote the sub-sequence $u_{j_1}\dots u_{j_n}$ such that $j_1<\ldots<j_n$ and $\{j_1,\dots,j_n\}=\{j\leq|\vec{z}|\mid z_j\in\mathcal{X}^\Box\}$.

Strictly positive positions. Let $f : (\vec{x} : \vec{T})U$ with $stat_f = lex\ \vec{m}$. The set of *strictly positive positions* of f, $SP(f)$, is defined as follows. Assume that $m_i = mul\ x_{k_1} \dots x_{k_p}$. Then, $i \in SP(f)$ iff there exist $T_f^i = C\vec{a}$ such that C is strictly positive and, for all j, $T_{k_j} = C\vec{u}$ with $C \in \mathcal{CF}^{\Box}$ and $\vec{u}|_C = \vec{a}|_C$.

Assignment compatibility. A status assignment *stat* is *compatible* with a precedence $\geq_{\mathcal{F}}$ if $f =_{\mathcal{F}} g$ implies $stat_f = stat_g$, $SP(f) = SP(g)$ and, for all $i \in SP(f)$, $T_f^i = T_g^i$.

Status ordering. Let $>$ be an ordering on terms and $stat = lex\ \vec{m}$ be a status of arity n. The *extension* of $>$ to the sequences of terms of length n is the ordering $>_{stat}$ defined as follows:

- $\vec{u} >_{stat} \vec{v}$ if $\vec{m}\{\vec{x} \mapsto \vec{u}\}\ (>^m)_{\text{lex}}\ \vec{m}\{\vec{x} \mapsto \vec{v}\}$,
- $mul\ \vec{u} >^m mul\ \vec{v}$ if $\{\vec{u}\} >_{\text{mul}} \{\vec{v}\}$.

For instance, if $stat = lex(mul\ x_2)(mul\ x_1\ x_3)$ then $(u_1, u_2, u_3) >_{stat} (v_1, v_2, v_3)$ iff $(\{u_2\}, \{u_1, u_3\})\ (>_{\text{mul}})_{\text{lex}}\ (\{v_2\}, \{v_1, v_3\})$. An important property of $>_{stat}$ is that it is well-founded whenever $>$ is.

We now define the computability closure of a rule $R = (l \to r, \Gamma, \rho)$ with $l = f\vec{l}$, $f : (\vec{x} : \vec{T})U$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$.

Definition 24 (Ordering on symbol arguments) The ordering $>_R$ on arguments of f is an adaptation of $>_{stat_f}$ where the ordering $>$ depends on the type (basic or strictly positive) of the argument. Assume that $stat_f = lex\ m_1 \dots m_k$. Then:

- $\vec{t} : \vec{T} >_R \vec{u} : \vec{U}$ if $\vec{m}\{\vec{x} \mapsto (\vec{t} : \vec{T})\}\ (>^1, \dots, >^k)_{\text{lex}}\ \vec{m}\{\vec{x} \mapsto (\vec{u} : \vec{U})\}$.
- $mul(\vec{t} : \vec{T}) >^i mul(\vec{u} : \vec{U})$ if $i \in SP(f)$ and $\{\vec{t} : \vec{T}\}\ (>_R^i)_{\text{mul}}\ \{\vec{u} : \vec{U}\}$,
- $mul(\vec{t} : \vec{T}) >^i mul(\vec{u} : \vec{U})$ if $i \notin SP(f)$ and $\{\vec{t} : \vec{T}\}\ (\rhd_\rho^+)_{\text{mul}}\ \{\vec{u} : \vec{U}\}$,
- $t : T\ >_R^i\ u : U$ if:
  - $t = f\vec{t}$, $f : (\vec{x} : \vec{T})C\vec{v}$, $\gamma = \{\vec{x} \mapsto \vec{t}\}$ and no $D =_{\mathcal{C}} C$ occurs in $\vec{v}\gamma\rho$,
  - $u = x\vec{u}$ with $x \in \text{dom}(\Gamma)$,
  - $t : T\ \rhd_\rho^+\ x : V$,
  - $V\rho = x\Gamma = (\vec{y} : \vec{U})C\vec{w}$, $\delta = \{\vec{y} \mapsto \vec{u}\}$, $U\rho = C\vec{w}\delta$ and no $D =_{\mathcal{C}} C$ occurs in $\vec{U}\delta$,
  - $\vec{v}\gamma\rho|_C = \vec{w}\delta|_C$.

One can easily check that, for the addition on ordinals, $lim\ f : ord >_R^1 fn : ord$. Indeed, for this rule, one can take $\Gamma = x : ord, f : nat \Rightarrow ord$ and the identity for ρ. Then, $f \in \text{dom}(\Gamma)$, $f\Gamma = nat \Rightarrow ord$ and $lim\ f : ord \rhd_\rho f : nat \Rightarrow ord$.

Definition 25 (Computability closure) Let $\mathcal{F}' = \mathcal{F} \cup \text{dom}(\Gamma)$, $\mathcal{X}' = \mathcal{X} \setminus \text{FV}(l)$, $\mathcal{T} = \mathcal{T}(\mathcal{F}', \mathcal{X}')$ and $\mathcal{E}' = \mathcal{E}(\mathcal{F}', \mathcal{X}')$. The *computability closure* of R w.r.t. a precedence $\geq_{\mathcal{F}}$ and a status assignment *stat* compatible with $\geq_{\mathcal{F}}$ is the smallest relation $\vdash_{\text{c}} \subseteq \mathcal{E}' \times \mathcal{T}' \times \mathcal{T}'$ defined by the inference rules of Figure 4 where, for all $x \in \text{dom}(\Gamma)$, $\tau_x = x\Gamma$ and $x <_{\mathcal{F}} f$, and where $\delta : \Gamma_g \rightsquigarrow_c \Delta$ means that, for all $y \in \text{dom}(\Gamma_g)$, $\Delta \vdash_{\text{c}} x\delta : x\Gamma_g\delta$.

Note that the computability closure can easily be extended by adding new inference rules. Then, for preserving strong normalization, it suffices to complete the proof of

Fig. 4. Computability closure of $R = (f\vec{l} \to r, \Gamma, \rho)$ with $f : (\vec{x} : \vec{T})U$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$

$$\text{(ax)}\quad \frac{}{\vdash_c \star : \Box}$$

$$\text{(symb}^<\text{)}\quad \frac{\vdash_c \tau_g : s_g}{\vdash_c g : \tau_g} \qquad (g <_{\mathcal{F}} f)$$

$$\text{(symb}^=\text{)}\quad \frac{\vdash_c \tau_g : s_g \quad \delta : \Gamma_g \rightsquigarrow_c \Delta}{\Delta \vdash_c g\vec{y}\delta : V\delta} \qquad \begin{array}{l}(g =_{\mathcal{F}} f,\ g : (\vec{y} : \vec{U})V,\\ \vec{y}\delta : \vec{U}\delta <_R \vec{x}\gamma : \vec{T}\gamma)\end{array}$$

$$\text{(var)}\quad \frac{\Delta \vdash_c T : s_x}{\Delta, x : T \vdash_c x : T} \qquad (x \notin \mathrm{dom}(\Delta))$$

$$\text{(weak)}\quad \frac{\Delta \vdash_c t : T \quad \Delta \vdash_c U : s_x}{\Delta, x : U \vdash_c t : T} \qquad (x \notin \mathrm{dom}(\Delta))$$

$$\text{(prod)}\quad \frac{\Delta, x : U \vdash_c V : s}{\Delta \vdash_c (x : U)V : s}$$

$$\text{(abs)}\quad \frac{\Delta, x : U \vdash_c v : V \quad \Delta \vdash_c (x : U)V : s}{\Delta \vdash_c [x : U]v : (x : U)V}$$

$$\text{(app)}\quad \frac{\Delta \vdash_c t : (x : U)V \quad \Delta \vdash_c u : U}{\Delta \vdash_c tu : V\{x \mapsto u\}}$$

$$\text{(conv)}\quad \frac{\Delta \vdash_c t : T \quad \Delta \vdash_c T : s \quad \Delta \vdash_c T' : s}{\Delta \vdash_c t : T'} \qquad (T \downarrow T')$$

Theorem 67 where we prove that the rules of the computability closure indeed preserve computability.

Definition 26 (Well-formed rule) R is *well-formed* if:
- $\Gamma \vdash l\rho : U\gamma\rho$,
- $\forall x \in \mathrm{dom}(\Gamma)$, $\exists i$, $l_i : T_i\gamma \ \rhd_\rho^* \ x : x\Gamma$,
- $\mathrm{dom}(\rho) \subseteq \mathrm{FV}(l) \setminus \mathrm{dom}(\Gamma)$.

For instance, consider the rule:

$$app\ p\ (cons\ x\ n\ \ell)\ n'\ \ell' \quad \to \quad cons\ x\ (n + n')\ (app\ n\ \ell\ n'\ \ell')$$

with $\Gamma = x : nat, n : nat, \ell : listn, n' : nat, \ell' : listn'$ and $\rho = \{p \mapsto sn\}$. We have $\Gamma \vdash l\rho : list(p + n')\rho$. For x, we have $cons\ x\ n\ \ell : listp \rhd_\rho x : nat$. One can easily check that the conditions are also satisfied for the other variables.

Definition 27 (Computable system) R satisfies the *General Schema* w.r.t. a precedence $\geq_{\mathcal{F}}$ and a status assignment *stat* compatible with $\geq_{\mathcal{F}}$ if it is well-formed and if

$\vdash_c r : U\gamma\rho$. A set of rules $\mathcal{R}$ is *computable* if there exists a precedence $\geq_{\mathcal{F}}$ and a status assignment *stat* compatible with $\geq_{\mathcal{F}}$ for which every rule of $\mathcal{R}$ satisfies the General Schema w.r.t. $\geq_{\mathcal{F}}$ and *stat*.

To summarize, the rule $(l \to r, \Gamma, \rho)$ is well-typed and satisfies the General Schema if:
- $\Gamma \vdash l\rho : U\gamma\rho$,
- $\forall \Delta, \sigma, T$, if $\Delta \vdash l\sigma : T$ then $\sigma : \Gamma \rightsquigarrow \Delta$ and $\sigma \downarrow \rho\sigma$,
- $\forall x \in \mathrm{dom}(\Gamma)$, $\exists i$, $l_i : T_i\gamma \ \rhd_\rho^* \ x : x\Gamma$,
- $\mathrm{dom}(\rho) \subseteq \mathrm{FV}(l) \setminus \mathrm{dom}(\Gamma)$,
- $\vdash_c r : U\gamma\rho$.

Because of the (conv) rule, the relation $\vdash_c$ may be undecidable. On the other hand, if we restrict the (conv) rule to a confluent and strongly normalizing fragment of $\to$, then $\vdash_c$ becomes decidable (with an algorithm similar to the one for $\vdash$). This is quite reasonable since, in practice, the symbols and the rules are often added one after the other (or by groups, but the argument can be generalized), thus confluence and strong normalization can be shown incrementally.

For instance, let $(\mathcal{F}, \mathcal{R})$ be a confluent and strongly normalizing system, $f \notin \mathcal{F}$ and $\mathcal{R}_f$ be a set of rules defining f and whose symbols belong to $\mathcal{F}' = \mathcal{F} \cup \{f\}$. Then, $(\mathcal{F}', \mathcal{R})$ is also confluent and strongly normalizing. Thus, we can check that the rules of $\mathcal{R}_f$ satisfy the General Schema with the rule (conv) restricted to the case where $T \downarrow_{\beta\mathcal{R}} T'$. This does not seem a big restriction: it would be surprising that the typing of a rule requires the use of the rule itself !

We now detail the case of $app\ p\ (cons\ x\ n\ \ell)\ n'\ \ell' \to cons\ x\ (n + n')\ (app\ n\ \ell\ n'\ \ell')$. We take $stat_{app} = lex(mul\ x_2)$; $app >_{\mathcal{F}} cons, +$; $cons >_{\mathcal{F}} nat$ and $+ >_{\mathcal{F}} s, 0 >_{\mathcal{F}} nat$. We have already seen that this rule is well-formed. Let us show that $\vdash_c r : list(sn)$.

For applying (symb$^<$), we must show that $\vdash_c \tau_{cons} : \star$, $\vdash_c x : nat$, $\vdash_c n + n' : nat$ and $\vdash_c app\ n\ \ell\ n'\ \ell' : list(n + n')$. The first assertions follow from the fact that the same judgements holds in $\vdash$ without using *app*. Hence, we are left to prove the last assertion.

For applying (symb$^=$), we must show that $\vdash_c \tau_{app} : \star$, $\vdash_c n : nat$, $\vdash_c \ell : listn$, $\vdash_c n' : nat$, $\vdash_c \ell' : listn'$ and $cons\ x\ n\ \ell : list(sn) \rhd_\rho \ell : listn$. The first assertions follow from the fact that the same judgements hold in $\vdash$ without using *app*. The last assertion has already been shown when proving that the rule is well-formed.

5.3. *Strong normalization conditions*

Definition 28 Let $\mathcal{G} \subseteq \mathcal{F}$. The *rewrite system* $(\mathcal{G}, \mathcal{R}_\mathcal{G})$ is:
- *first-order* if every rule of $\mathcal{R}_\mathcal{G}$ has an algebraic right hand-side and, for all $g \in \mathcal{G}$, either $g \in \mathcal{F}^\square$ or $g : (\vec{x} : \vec{T})C\vec{v}$ with $C \in \mathcal{C}\mathcal{F}^\square$ primitive.
- *primitive* if all the rules of $\mathcal{R}_\mathcal{G}$ have a right hand-side of the form $[\vec{x} : \vec{T}]g\vec{u}$ with g a symbol of $\mathcal{G}$ or a primitive constant predicate symbol.
- *simple* if there is no critical pair between $\mathcal{R}_\mathcal{G}$ and $\mathcal{R}$.
- *small* if, for every rule $g\vec{l} \to r \in \mathcal{R}_\mathcal{G}$, $\forall x \in \mathrm{FV}^\square(r)$, $\exists \kappa_x$, $l_{\kappa_x} = x$.
- *positive* if, for every $g \in \mathcal{G}$, for every rule $l \to r \in \mathcal{R}_\mathcal{G}$, $\mathrm{Pos}(g, r) \subseteq \mathrm{Pos}^+(r)$.

- *safe* if for every rule $(g\vec{l} \to r, \Gamma, \rho) \in \mathcal{R}_\mathcal{G}$ with $g : (\vec{x} : \vec{T})U$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$:
  - $\forall x \in \mathrm{FV}^\square(\vec{T}U)$, $x\gamma\rho \in \mathrm{dom}^\square(\Gamma)$,
  - $\forall x, x' \in \mathrm{FV}^\square(\vec{T}U)$, $x\gamma\rho = x'\gamma\rho \Rightarrow x = x'$.‡‡

Definition 29 (Strong normalization conditions)

(A0) All the rules are well-typed.

(A1) The relation $\to = \to_\mathcal{R} \cup \to_\beta$ is confluent on $\mathcal{T}$.

(A2) There exists an admissible inductive structure.

(A3) There exists a precedence $\succeq$ on $\mathcal{DF}^\square$ which is compatible with $\mathcal{R}_{\mathcal{DF}^\square}$ and whose equivalence classes form a system which is either:

- **(p)** primitive,
- **(q)** positive, small and simple,
- **(r)** computable, small and simple.

(A4) There exists a partition $\mathcal{F}_1 \uplus \mathcal{F}_\omega$ of $\mathcal{DF}$ (*first-order* and *higher-order* symbols) such that:

- **(a)** $(\mathcal{F}_\omega, \mathcal{R}_\omega)$ is computable,
- **(b)** $(\mathcal{F}_\omega, \mathcal{R}_\omega)$ is safe,
- **(c)** no symbol of $\mathcal{F}_\omega$ occurs in the rules of $\mathcal{R}_1$,
- **(d)** $(\mathcal{F}_1, \mathcal{R}_1)$ is first-order,
- **(e)** if $\mathcal{R}_\omega \neq \emptyset$ then $(\mathcal{F}_1, \mathcal{R}_1)$ is non-duplicating,
- **(f)** $\to_{\mathcal{R}_1}$ is strongly normalizing on first-order algebraic terms.

The condition (A1) ensures, among other things, that β preserves typing. This condition may seem difficult to fulfill since confluence is often proved by using strong normalization and local confluence of critical pairs (Nipkow 1991).

We know that $\to_\beta$ is confluent and that there is no critical pair between $\mathcal{R}$ and β since the left hand-sides of rules are algebraic. Müller (Müller 1992) showed that, in this case, if $\to_\mathcal{R}$ is confluent and all the rules of $\mathcal{R}$ are left-linear, then $\to_\mathcal{R} \cup \to_\beta$ is confluent. Thus, the possibility we have introduced of linearizing some rules (substitution ρ) appears to be very useful (see Definition 3).

In the case of left-linear rules, and assuming that $\to_{\mathcal{R}_1}$ is strongly normalizing as it is required in (f), how can we prove that $\to$ is confluent? In the case where $\to_{\mathcal{R}_1}$ is non-duplicating if $\mathcal{R}_\omega \neq \emptyset$, we show in Theorem 64 that $\to_{\mathcal{R}_1} \cup \to_{\mathcal{R}_\omega}$ is strongly normalizing. Therefore, it suffices to check that the critical pairs of $\mathcal{R}$ are confluent (without using any β-reduction).

In (A4), in the case where $\mathcal{R}_\omega \neq \emptyset$, we require that the rules of $\mathcal{R}_1$ are non-duplicating. Indeed, strong normalization is not a modular property (Toyama 1987), even with confluent systems (Drosten 1989). On the other hand, strong normalization is modular for

‡‡ All this means that $\gamma\rho$ is an injection from $\mathrm{FV}^\square(\vec{T}U)$ to $\mathrm{dom}^\square(\Gamma)$.

disjoint and non duplicating systems (Rusinowitch 1987). Here, $\mathcal{R}_1$ and $\mathcal{R}_\omega$ are not disjoint but hierarchically defined: by (c), no symbol of $\mathcal{F}_\omega$ occurs in the rules of $\mathcal{R}_1$. In (Dershowitz 1994), Dershowitz gathers some results on the modularity of strong normalization for first-order rewrite systems. It would be very interesting to study the modularity of strong normalization in the case of higher-order rewriting and, in particular, other conditions than non-duplication which, for example, does not allow us to accept the following definition:

$$\begin{array}{rcl} 0/y & \to & 0 \\ (s\ x)/y & \to & s((x-y)/y) \\ \\ 0-y & \to & 0 \\ (s\ x)-0 & \to & s\ x \\ (s\ x)-(s\ y) & \to & x-y \end{array}$$

This system is a duplicating first-order system not satisfying the General Schema: it can be put neither in $\mathcal{R}_1$ nor in $\mathcal{R}_\omega$. Note that Giménez (Giménez 1998) has developped a termination criterion for the Calculus of Inductive Constructions that accepts this example.

In (A3), the smallness condition for computable and positive systems is equivalent in the Calculus of Inductive Constructions to the restriction of strong elimination to "small" inductive types, that is, to the types whose constructors have no other predicate parameters than the ones of the type. For example, the type *list* of polymorphic list is small since, in the type $(A:\star)A \Rightarrow listA \Rightarrow listA$ of its constructor *cons*, A is a parameter of *list*. On the other hand, a type T having a constructor c of type $\star \Rightarrow T$ is not small. So, we cannot define a function f of type $T \Rightarrow \star$ with the rule $f(c\ A) \to A$. Such a rule is not small and does not form a primitive system either. In some sense, primitive systems can always be considered as small systems since they contain no projection and primitive predicate symbols have no predicate argument. This restriction is not only technical: elimination on big inductive types may lead to logical inconsistencies (Coquand 1986).

Finally, in (A4), the safeness condition for higher-order symbols means that one cannot do matching or equality tests on predicate arguments that are necessary for typing other arguments. In her extension of HORPO (Jouannaud and Rubio 1999) to the Calculus of Constructions, Walukiewicz (Walukiewicz-Chrząszcz 2002) requires a similar condition. This has to be related to the fact that the polymorphism of CC is essentially parametric, that is, a polymorphic function uses the same algorithm at all types (Reynolds 1983). Girard already demonstrated in (Girard 1971) that normalization fails if a non-parametric operator $J : (A:\star)(B:\star)A \Rightarrow B$ defined by $J\ A\ A\ x \to x$ is added to system F. See (Harper and Mitchell 1999) for an analysis of Girard's J operator. On the other hand, the rule $map\ A\ A\ [x:A]x\ \ell \to \ell$, which does not seem problematic, does not satisfy the safeness condition either (note however that the left hand-side if not algebraic).

We can now state our main result whose proof is the subject of Section 6:

THEOREM: If a CAC satisfies the conditions of Definition 29 then its reduction relation $\rightarrow = \rightarrow_{\mathcal{R}} \cup \rightarrow_\beta$ preserves typing and is strongly normalizing.

In (Blanqui 2001), we prove that most of CIC can be encoded into a CAC satisfying our conditions, and that our conditions can also be applied to prove the cut-elimination property in Natural Deduction Modulo (Dowek and Werner 1998). But let us give a more concrete example:

$$\begin{array}{lcl@{\qquad}lcl@{\qquad}lcl}
\neg\top & \rightarrow & \bot & P \vee \top & \rightarrow & \top & P \wedge \top & \rightarrow & P \\
\neg\bot & \rightarrow & \top & P \vee \bot & \rightarrow & P & P \wedge \bot & \rightarrow & \bot
\end{array}$$

$$\begin{array}{rcl@{\qquad}rcl}
x + 0 & \rightarrow & x & x \times 0 & \rightarrow & 0 \\
0 + x & \rightarrow & x & 0 \times x & \rightarrow & 0 \\
x + (s\ y) & \rightarrow & s(x + y) & x \times (s\ y) & \rightarrow & (x \times y) + x \\
(s\ x) + y & \rightarrow & s(x + y) & (s\ 0) \times x & \rightarrow & x \\
(x + y) + z & \rightarrow & x + (y + z) & x \times (s\ 0) & \rightarrow & x
\end{array}$$

$$\begin{array}{rcl}
eq\ A\ 0\ 0 & \rightarrow & \top \\
eq\ A\ 0\ (s\ x) & \rightarrow & \bot \\
eq\ A\ (s\ x)\ 0 & \rightarrow & \bot \\
eq\ A\ (s\ x)\ (s\ y) & \rightarrow & eq\ nat\ x\ y \\
app\ A\ (nil\ A')\ \ell & \rightarrow & \ell \\
app\ A\ (cons\ A'\ x\ \ell)\ \ell' & \rightarrow & cons\ A\ x\ (app\ A\ \ell\ \ell') \\
app\ A\ (app\ A'\ \ell\ \ell')\ \ell'' & \rightarrow & app\ A\ \ell\ (app\ A\ \ell'\ \ell'')
\end{array}$$

$$\begin{array}{rcl}
len\ A\ (nil\ A') & \rightarrow & 0 \\
len\ A\ (cons\ A'\ x\ \ell) & \rightarrow & s\ (len\ A\ \ell) \\
len\ A\ (app\ A'\ \ell\ \ell') & \rightarrow & (len\ A\ \ell) + (len\ A\ \ell')
\end{array}$$

$$\begin{array}{rcl}
in\ A\ x\ (nil\ A') & \rightarrow & \bot \\
in\ A\ x\ (cons\ A'\ y\ l) & \rightarrow & (eq\ A\ x\ y) \vee (in\ A\ x\ l)
\end{array}$$

$$\begin{array}{rcl}
incl\ A\ (nil\ A')\ l & \rightarrow & \top \\
incl\ A\ (cons\ A'\ x\ l)\ l' & \rightarrow & (in\ A\ x\ l') \wedge (incl\ A\ l\ l')
\end{array}$$

$$\begin{array}{rcl}
sub\ A\ (nil\ A')\ l & \rightarrow & \top \\
sub\ A\ (cons\ A'\ x\ l)\ (nil\ A'') & \rightarrow & \bot \\
sub\ A\ (cons\ A'\ x\ l)\ (cons\ A''\ x'\ l') & \rightarrow & ((eq\ A\ x\ x') \wedge (sub\ A\ l\ l')) \\
 & & \vee (sub\ A\ (cons\ A\ x\ l)\ l')
\end{array}$$

$$\begin{array}{rcl}
eq\ L\ (nil\ A)\ (nil\ A') & \rightarrow & \top \\
eq\ L\ (nil\ A)\ (cons\ A'\ x\ l) & \rightarrow & \bot \\
eq\ L\ (cons\ A'\ x\ l)\ (nil\ A) & \rightarrow & \bot \\
eq\ L\ (cons\ A\ x\ l)\ (cons\ A'\ x'\ l') & \rightarrow & (eq\ A\ x\ x') \wedge (eq\ (list\ A)\ l\ l')
\end{array}$$

This rewriting system is computable, simple, small, safe and confluent (this can be automatically proved by CiME (Contejean *et al.* 2000)). Since the rules are left-linear, the combination with $\rightarrow_\beta$ is also confluent. Therefore, the conditions of strong normalization are satisfied. For example, for the last rule, take $\Gamma = A : \star, x : A, x' : A, \ell : listA, \ell' : listA$

and $\rho = \{A' \mapsto A, L \mapsto listA\}$. The rule is well-formed ($cons(A', x', \ell') : L \ \triangleright_\rho \ x' : A'$, ...) and satisfies the General Schema ($\{cons(A, x, \ell) : L, cons(A', x', \ell') : L\}$ $(\triangleright_\rho)_{\mathrm{mul}}$ $\{x : A, x' : A'\}$ and $\{\ell : listA, \ell' : listA\}$).

However, the system lacks several important rules to get a complete decision procedure for classical propositional tautologies (Figure 1 in Section 1) or other simplification rules on the equality (Figure 2 in Section 1). To accept these rules, we must consider rewriting modulo associativity and commutativity and get rid of the simplicity conditions. Moreover, the distributivity rule $P \wedge (Q \oplus R) \to (P \wedge Q) \oplus (P \wedge R)$ is not small. Rewriting modulo AC does not seem to be a difficult extension, except perhaps in the case of predicate-level rewriting. On the other hand, confluence, simplicity and smallness seem difficult problems.

From strong normalization, we can deduce the decidability of the typing relation, which is the essential property on which proof assistants like Coq (Coq Development Team 2002) or LEGO (Luo and Pollack 1992) are based.

Theorem 30 (Decidability of type-checking) Let Γ be a valid environment and T be $\Box$ or a term typable in Γ. In a CAC satisfying the conditions of Definition 29, checking whether a term t is of type T in Γ is decidable.

Proof. Since Γ is valid, it is possible to say whether t is typable and, if so, it is possible to infer a type T' for t. Since types are convertible, it suffices to check that T and T' have the same normal form. The reader is invited to look at (Coquand 1991; Barras 1999) for more details. □

6. Correctness of the conditions

Our strong normalization proof is based on Tait and Girard's method of computability predicates and reducibility candidates (Girard *et al.* 1988). The idea is to interpret each type T as a set $[\![T]\!]$ of strongly normalizable terms and to prove that every term of type T belongs to $[\![T]\!]$. The reader not familiar with these notions is invited to read the Chapter 3 of the Ph.D. thesis of Werner (Werner 1994) for an introduction, and the paper of Gallier for a more detailed presentation (Gallier 1990).

It is worth noting several differences with previous strong normalization proofs:

- The present proof is an important simplification of the proof given in (Blanqui 2001), which uses candidates *à la* Coquand and Gallier (Coquand and Gallier 1990) where only well-typed terms are considered. Here, candidates are made of well-typed and not well-typed terms. This leads to simpler notations and less properties to be care of.
- In (Geuvers 1994), Geuvers uses candidates with possibly not well-typed terms too. However, the way dependent types are interpreted does not allow this proof to be extended to type-level rewriting. Indeed, in this proof, dependencies are simply ignored but, if one has a predicate symbol $F : nat \Rightarrow \star$ defined by $F0 \to nat$ and $F(sn) \to nat \Rightarrow nat$, then one expects $F0$ to be interpreted as nat, and $F(sn)$ as $nat \Rightarrow nat$.
- In (Werner 1994), Werner uses candidates with (not well-typed) pure λ-terms, that

is, terms without type annotation in abstractions, in order to deal with η-conversion, whose combination with β is not confluent on annotated terms. As a consequence, he has to define a translation from annotated terms to pure terms that implies the strong normalization of annotated terms. Here, we give a direct proof.

6.1. *Reducibility candidates*

We denote by:
- $\mathcal{SN}$ the set of strongly normalizable terms,
- $\mathcal{WN}$ the set of weakly normalizable terms,
- $\mathcal{CR}$ the set of terms from which reductions are confluent.

Definition 31 (Neutral terms) A term t is *neutral* if it is not of the following form:
- abstraction: $[x:T]u$,
- partial application: $f\vec{t}$ with $f \in \mathcal{DF}$ and $|\vec{t}| < |\vec{l}|$ for some rule $f\vec{l} \to r \in \mathcal{R}$,
- constructor: $f\vec{t}$ with $f : (\vec{x}:\vec{T})C\vec{v}$ and $C \in \mathcal{CF}^{\square}$.

Let $\mathcal{N}$ be the set of neutral terms.

Note that, if t is neutral, then tu is neutral and not head-reducible.

Definition 32 (Reducibility candidates) We inductively define the set $\mathcal{R}_t$ of the interpretations for the terms of type t, the ordering $\leq_t$ on $\mathcal{R}_t$, the element $\top_t \in \mathcal{R}_t$, and the operation $\bigwedge_t$ from the powerset of $\mathcal{R}_t$ to $\mathcal{R}_t$ as follows. If $t \neq \square$ and $\Gamma \not\vdash t : \square$ then:
- $\mathcal{R}_t = \{\emptyset\}$, $\leq_t = \subseteq$, $\top_t = \emptyset$ and $\bigwedge_t(\Re) = \top_t$.

Otherwise:
- $\mathcal{R}_s$ is the set of all the subsets R of $\mathcal{T}$ such that:
  - **(R1)** $R \subseteq \mathcal{SN}$ (strong normalization).
  - **(R2)** If $t \in R$ then $\to(t) \subseteq R$ (stability by reduction).
  - **(R3)** If $t \in \mathcal{N}$ and $\to(t) \subseteq R$ then $t \in R$ (neutral terms).

  Furthermore, $\leq_s = \subseteq$, $\top_s = \mathcal{SN}$, $\bigwedge_s(\Re) = \bigcap \Re$ if $\Re \neq \emptyset$, and $\bigwedge_s(\emptyset) = \top_s$.
- $\mathcal{R}_{(x:U)K}$ is the set of functions R from $\mathcal{T} \times \mathcal{R}_U$ to $\mathcal{R}_K$ such that $R(u,S) = R(u',S)$ whenever $u \to u'$, $\top_{(x:U)K}(u,S) = \top_K$, $\bigwedge_{(x:U)K}(\Re)(u,S) = \bigwedge_K(\{R(u,S) \mid R \in \Re\})$, and $R \leq_{(x:U)K} R'$ iff, for all (u,S), $R(u,S) \leq_K R'(u,S)$.

Lemma 33 $\mathcal{V} = \{x\vec{t} \in \mathcal{T} \mid x \in \mathcal{X}, \vec{t} \in \mathcal{SN}\} \neq \emptyset$ and, for all $R \in \mathcal{R}_s$, $\mathcal{V} \subseteq R$.

Proof. $\mathcal{V} \neq \emptyset$ since $\mathcal{X} \neq \emptyset$. Let $R \in \mathcal{R}_s$. We prove that $x\vec{t} \in R$ by induction on $\vec{t}$ with $\to_{\text{lex}}$ as well-founded ordering ($\vec{t} \in \mathcal{SN}$). Since $x\vec{t} \in \mathcal{N}$, it suffices to prove that $\to(x\vec{t}) \subseteq R$, which is the induction hypothesis. □

Lemma 34 (a) If $T\ \mathbb{C}^*_\Gamma\ T'$ then $\mathcal{R}_T = \mathcal{R}_{T'}$.
(b) If $\Gamma \vdash T : s$ and $\theta : \Gamma \rightsquigarrow \Delta$ then $\mathcal{R}_T = \mathcal{R}_{T\theta}$.

Proof.

(a) By induction on the size of T. If $\Gamma \vdash T : \star$ then $\Gamma \vdash T' : \star$ and $\mathcal{R}_T = \{\emptyset\} = \mathcal{R}_{T'}$. Assume now that $\Gamma \vdash T : \Box$. If $T = \star$ then $T' = \star$ and $\mathcal{R}_T = \mathcal{R}_{T'}$. If $T = (x : U)K$ then $T' = (x : U')K'$ with $U\ \mathbb{C}^*_\Gamma\ U'$ and $K\ \mathbb{C}^*_{\Gamma,x:U}\ K'$. By induction hypothesis, $\mathcal{R}_U = \mathcal{R}_{U'}$ and $\mathcal{R}_K = \mathcal{R}_{K'}$. Therefore, $\mathcal{R}_T = \mathcal{R}_{T'}$.

(b) By induction on the size of T. If $\Gamma \vdash T : \star$ then $\Delta \vdash T\theta : \star$ and $\mathcal{R}_T = \{\emptyset\} = \mathcal{R}_{T\theta}$. Assume now that $\Gamma \vdash T : \Box$. If $T = \star$, this is immediate. If $T = (x : U)K$ then $T\theta = (x : U\theta)K\theta$. By induction hypothesis, $\mathcal{R}_U = \mathcal{R}_{U\theta}$ and $\mathcal{R}_K = \mathcal{R}_{K\theta}$. Therefore, $\mathcal{R}_T = \mathcal{R}_{T\theta}$.

□

Lemma 35 (Completeness of the candidates lattice) $(\mathcal{R}_t, \leq_t)$ is a complete lattice with greatest element $\top_t$ and the lower bound of $\Re \subseteq \mathcal{R}_t$ given by $\bigwedge_t(\Re)$.

Proof. It suffices to prove that $(\mathcal{R}_t, \leq_t)$ is a complete inf-semi-lattice and that $\top_t$ is its greatest element. One can easily check by induction on t that $\leq_t$ is an ordering (*i.e.* is reflexive, transitive and anti-symmetric), $\top_t$ is the greatest element of $\mathcal{R}_t$, and $\bigwedge_t(\Re)$ is the lower bound of $\Re \subseteq \mathcal{R}_t$. □

Lemma 36 (Smallest element) Let $\bot_0 = \emptyset$ and $\bot_{i+1} = \bot_i \cup \{t \in \mathcal{N} \mid \to(t) \subseteq \bot_i\}$. The set $\bot_s = \bigcup\{\bot_i \mid i < \omega\}$ is the smallest element of $\mathcal{R}_s$: $\bot_s = \bigcap \mathcal{R}_s$.

Proof. Let $R \in \mathcal{R}_s$. We prove by induction on i that $\bot_i \subseteq R$. For $i = 0$, this is immediate. Assume that $\bot_i \subseteq R$ and let $t \in \bot_{i+1} \setminus \bot_i$. We have $t \in \mathcal{N}$ and $\to(t) \subseteq R$ by induction hypothesis. Therefore, by (R3), $t \in R$ and $\bot_s \subseteq R$ for all $R \in \mathcal{R}_s$. Thus, $\bot_s \subseteq \bigcap \mathcal{R}_s$.

We now prove that $\bot_s \in \mathcal{R}_s$, hence that $\bot_s = \bigcap \mathcal{R}_s$.

(R1) We prove that $\bot_i \subseteq \mathcal{SN}$ by induction on i. For $i = 0$, this is immediate. Assume that $\bot_i \subseteq \mathcal{SN}$ and let $t \in \bot_{i+1} \setminus \bot_i$. We have $\to(t) \subseteq \mathcal{SN}$ by induction hypothesis. Therefore, $t \in \mathcal{SN}$.

(R2) Let $t \in \bot_s$. Since $\bot_0 = \emptyset$, $t \in \bot_{i+1} \setminus \bot_i$ for some i. So, $\to(t) \subseteq \bot_i \subseteq \bot_s$.

(R3) Let $t \in \mathcal{N}$ with $\to(t) \subseteq \bot_s$. Since $\to$ is assumed to be finitely branching, $\to(t) = \{t_1, \ldots, t_n\}$. For all i, there exists k_i such that $t_i \in \bot_{k_i}$. Let k be the max of $\{k_1, \ldots, k_n\}$. We have $\to(t) \subseteq \bot_k$ and thus $t \in \bot_{k+1} \subseteq \bot_s$.

□

6.2. *Interpretation schema*

The interpretation $[\![t]\!]$ of a term t is defined by using a *candidate assignment* ξ for the free variables and an interpretation I for the predicate symbols. The interpretation of constant predicate symbols will de defined in Section 6.3, and the interpretation of defined predicate symbols in Section 6.5.

Definition 37 (Interpretation schema) A *candidate assignment* is a function ξ from $\mathcal{X}$ to $\bigcup\{\mathcal{R}_t \mid t \in \mathcal{T}\}$. A candidate assignment ξ *validates* an environment Γ or is a

Γ-*assignment*, written $\xi \models \Gamma$, if, for all $x \in \mathrm{dom}(\Gamma)$, $x\xi \in \mathcal{R}_{x\Gamma}$. An *interpretation* of a symbol f is an element of $\mathcal{R}_{\tau_f}$. An *interpretation* of a set $\mathcal{G}$ of symbols is a function which, to each symbol $g \in \mathcal{G}$, associates an interpretation of g.

The *interpretation* of t w.r.t. a candidate assignment ξ, an interpretation I and a substitution θ, is defined by induction on t as follows:

- $[\![t]\!]^I_{\xi,\theta} = \top_t$ if t is an object or a sort,
- $[\![f]\!]^I_{\xi,\theta} = I_f$,
- $[\![x]\!]^I_{\xi,\theta} = x\xi$,
- $[\![(x:U)V]\!]^I_{\xi,\theta} = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]^I_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^I_{\xi^S_x,\theta^u_x}\}$,
- $[\![[x:U]v]\!]^I_{\xi,\theta}(u,S) = [\![v]\!]^I_{\xi^S_x,\theta^u_x}$,
- $[\![tu]\!]^I_{\xi,\theta} = [\![t]\!]^I_{\xi,\theta}(u\theta, [\![u]\!]^I_{\xi,\theta})$,

where $\theta^u_x = \theta \cup \{x \mapsto u\}$ and $\xi^S_x = \xi \cup \{x \mapsto S\}$. In the case where $\Gamma \vdash t : s$, the elements of $[\![t]\!]^I_{\xi,\theta}$ are called *computable*. A substitution θ is *adapted* to a Γ-assignment ξ if $\mathrm{dom}(\theta) \subseteq \mathrm{dom}(\Gamma)$ and, for all $x \in \mathrm{dom}(\theta)$, $x\theta \in [\![x\Gamma]\!]^I_{\xi,\theta}$. A pair (ξ,θ) is Γ-*valid*, written $\xi,\theta \models \Gamma$, if $\xi \models \Gamma$ and θ is adapted to ξ.

After Lemma 33, the identity substitution is adapted to any Γ-candidate assignment.

Lemma 38 (Correctness of the interpretation schema) If $\Gamma \vdash t : T$ and $\xi \models \Gamma$ then $[\![t]\!]^I_{\xi,\theta} \in \mathcal{R}_T$. Moreover, if $\theta \to \theta'$ then $[\![t]\!]^I_{\xi,\theta} = [\![t]\!]^I_{\xi,\theta'}$.

Proof. By induction on $\Gamma \vdash t : T$.

(ax) $[\![\star]\!]^I_{\xi,\theta} = \top_\star = \mathcal{SN} \in \mathcal{R}_\Box$ and $[\![\star]\!]^I_{\xi,\theta}$ does not depend on θ.

(symb) $[\![f]\!]^I_{\xi,\theta} = I_f \in \mathcal{R}_{\tau_f}$ by assumption on I and $[\![f]\!]^I_{\xi,\theta}$ does not depend on θ.

(var) $[\![x]\!]^I_{\xi,\theta}$ does not depend on θ. Now, if $x \in \mathcal{X}^\star$ then $[\![x]\!]^I_{\xi,\theta} = \emptyset \in \mathcal{R}_T = \{\emptyset\}$. Otherwise, $[\![x]\!]^I_{\xi,\theta} = x\xi \in \mathcal{R}_T$ since $\xi \models \Gamma, x : T$.

(weak) By induction hypothesis.

(prod) $R = [\![(x:U)V]\!]^I_{\xi,\theta} = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]^I_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^I_{\xi^S_x,\theta^u_x}\} \in \mathcal{R}_s$ if it satisfies the properties (R1) to (R3):

- **(R1)** Strong normalization. Let $t \in R$. By induction hypothesis, $[\![U]\!]^I_{\xi,\theta} \in \mathcal{R}_{s'}$ for some s', and $[\![V]\!]^I_{\xi^S_x,\theta^u_x} \in \mathcal{R}_s$. Therefore, $\mathcal{X} \subseteq [\![U]\!]^I_{\xi,\theta}$ and $[\![V]\!]^I_{\xi^S_x,\theta^u_x} \subseteq \mathcal{SN}$. Take $u = x \in \mathcal{X}$. Then, $tx \in [\![V]\!]^I_{\xi^S_x,\theta}$ and $t \in \mathcal{SN}$.
- **(R2)** Stability by reduction. Let $t \in R$ and $t' \in {\to}(t)$. Let $u \in [\![U]\!]^I_{\xi,\theta}$ and $S \in \mathcal{R}_U$. Then, $tu \in [\![V]\!]^I_{\xi^S_x,\theta^u_x}$ which, by induction hypothesis, is stable by reduction. Therefore, since $t'u \in {\to}(tu)$, $t'u \in [\![V]\!]^I_{\xi^S_x,\theta^u_x}$ and $t' \in R$.
- **(R3)** Neutral terms. Let t be a neutral term such that ${\to}(t) \subseteq R$. Let $u \in [\![U]\!]^I_{\xi,\theta}$ and $S \in \mathcal{R}_U$. Since t is neutral, tu is neutral and, by induction hypothesis, $tu \in R' = [\![V]\!]^I_{\xi^S_x,\theta^u_x}$ if ${\to}(tu) \subseteq R'$. We prove it by induction on u with $\to$ as well-founded ordering ($u \in \mathcal{SN}$ by induction hypothesis). Since t is neutral, tu is not head-reducible and a reduct of tu is either of the form $t'u$ with $t' \in {\to}(t)$, or of the form tu' with $u' \in {\to}(u)$. In the former case, $t'u \in R'$ by assumption. In the latter case, we conclude by induction hypothesis.

Assume now that $\theta \to \theta'$. Let $R' = [\![(x:U)V]\!]^I_{\xi,\theta'} = \{t \in \mathcal{T} | \forall u \in [\![U]\!]^I_{\xi,\theta'}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^I_{\xi^S_x,\theta'^u_x}\}$. By induction hypothesis, $[\![U]\!]^I_{\xi,\theta'} = [\![U]\!]^I_{\xi,\theta}$ and $[\![V]\!]^I_{\xi^S_x,\theta'^u_x} = [\![V]\!]^I_{\xi^S_x,\theta^u_x}$. Therefore, $R' = R$.

(abs) Let $R = [\![[x:U]v]\!]^I_{\xi,\theta}$. $R(u,S) = [\![v]\!]^I_{\xi^S_x,\theta^u_x}$. By induction hypothesis, $R(u,S) \in \mathcal{R}_V$. Assume now that $u \to u'$. Then, $R(u',S) = [\![v]\!]^I_{\xi^S_x,\theta^{u'}_x}$. By induction hypothesis, $[\![v]\!]^I_{\xi^S_x,\theta^{u'}_x} = [\![v]\!]^I_{\xi^S_x,\theta^u_x}$. Therefore, $R \in \mathcal{R}_{(x:U)V}$. Assume now that $\theta \to \theta'$. Let $R' = [\![[x:U]v]\!]^I_{\xi,\theta'}$. $R'(u,S) = [\![v]\!]^I_{\xi^S_x,\theta'^u_x}$. By induction hypothesis, $R'(u,S) = R(u,S)$. Therefore, $R = R'$.

(app) Let $R = [\![tu]\!]^I_{\xi,\theta} = [\![t]\!]^I_{\xi,\theta}(u\theta, [\![u]\!]^I_{\xi,\theta})$. By induction hypothesis, $[\![t]\!]^I_{\xi,\theta} \in \mathcal{R}_{(x:U)V}$ and $[\![u]\!]^I_{\xi,\theta} \in \mathcal{R}_U$. Therefore, $[\![tu]\!]^I_{\xi,\theta} \in \mathcal{R}_V = \mathcal{R}_{V\{x\mapsto u\}}$ by Lemma 34. Assume now that $\theta \to \theta'$. Then, $R' = [\![tu]\!]^I_{\xi,\theta'} = [\![t]\!]^I_{\xi,\theta'}(u\theta', [\![u]\!]^I_{\xi,\theta'})$. By induction hypothesis, $[\![t]\!]^I_{\xi,\theta'} = [\![t]\!]^I_{\xi,\theta}$ and $[\![u]\!]^I_{\xi,\theta'} = [\![u]\!]^I_{\xi,\theta}$. Finally, since $[\![t]\!]^I_{\xi,\theta}$ is stable by reduction and $u\theta \to^* u\theta'$, we have $R = R'$.

(conv) By induction hypothesis since, by Lemma 34, $\mathcal{R}_T = \mathcal{R}_{T'}$.

□

Lemma 39 Let I and I' be two interpretations equal on the predicate symbols occurring in t, ξ and ξ' be two candidate assignments equal on the predicate variables free in t, and θ and θ' be two substitutions equal on the variables free in t. If $\Gamma \vdash t : T$ and $\xi \models \Gamma$ then $[\![t]\!]^{I'}_{\xi',\theta'} = [\![t]\!]^I_{\xi,\theta}$.

Proof. By induction on t. □

Lemma 40 (Candidate substitution) If $\Gamma \vdash t : T$, $\sigma : \Gamma \rightsquigarrow \Delta$ and $\xi \models \Delta$ then, for all θ, $[\![t\sigma]\!]^I_{\xi,\theta} = [\![t]\!]^I_{\xi',\sigma\theta}$ with $x\xi' = [\![x\sigma]\!]^I_{\xi,\theta}$ and $\xi' \models \Gamma$.

Proof. We first check that $\xi' \models \Gamma$. Let $x \in \mathrm{dom}(\Gamma)$. $x\xi' = [\![x\sigma]\!]_{\xi,\theta}$. By Lemma 38, $x\xi' \in \mathcal{R}_{x\Gamma\sigma}$ since $\Delta \vdash x\sigma : x\Gamma\sigma$ and $\xi \models \Delta$. By Lemma 34, $\mathcal{R}_{x\Gamma\sigma} = \mathcal{R}_{x\Gamma}$ since $\Gamma \vdash x\Gamma : s_x$ and $\sigma : \Gamma \rightsquigarrow \Delta$. We now prove the lemma by induction on t. If t is an object then $t\sigma$ is an object too and $[\![t\sigma]\!]^I_{\xi,\theta} = \emptyset = [\![t]\!]^I_{\xi',\sigma\theta}$. If t is not an object then $t\sigma$ is not an object either. We proceed by case on t:

- $[\![s\sigma]\!]^I_{\xi,\theta} = \top_s = [\![s]\!]^I_{\xi',\sigma\theta}$.
- $[\![f\sigma]\!]^I_{\xi,\theta} = I_f = [\![f]\!]^I_{\xi',\sigma\theta}$.
- $[\![x\sigma]\!]^I_{\xi,\theta} = x\xi' = [\![x]\!]^I_{\xi',\sigma\theta}$.
- Let $R = [\![(x:U\sigma)V\sigma]\!]^I_{\xi,\theta} = \{t \in \mathcal{T} \mid \forall u \in [\![U\sigma]\!]^I_{\xi,\theta}, \forall S \in \mathcal{R}_{U\sigma} = \mathcal{R}_U,\ tu \in [\![V\sigma]\!]^I_{\xi^S_x,\theta^u_x}\}$ and $R' = [\![(x:U)V]\!]^I_{\xi',\sigma\theta} = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]^I_{\xi',\sigma\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^I_{\xi'^S_x,(\sigma\theta)^u_x}\}$. By induction hypothesis, $[\![U\sigma]\!]^I_{\xi,\theta} = [\![U]\!]^I_{\xi',\sigma\theta}$ and $[\![V\sigma]\!]^I_{\xi^S_x,\theta^u_x} = [\![V]\!]^I_{\xi'',\sigma(\theta^u_x)}$ with $y\xi'' = [\![y\sigma]\!]_{\xi^S_x,\theta^u_x}$. Since $\sigma(\theta^u_x) = (\sigma\theta)^u_x$ ($x \notin \mathrm{dom}(\sigma) \cup \mathrm{dom}(\theta) \cup \mathrm{FV}(\sigma)$) and $\xi'' = \xi'^S_x$ ($x \notin \mathrm{dom}(\sigma) \cup \mathrm{FV}(\sigma)$), we have $R = R'$.
- Let $R = [\![[x:U\sigma]v\sigma]\!]^I_{\xi,\theta}$ and $R' = [\![[x:U]v]\!]^I_{\xi',\sigma\theta}$. By Lemma 34, R and R' have the same domain $\mathcal{T} \times \mathcal{R}_U$ and the same codomain $\mathcal{R}_V$. Moreover, $R(u,S) = [\![v\sigma]\!]^I_{\xi^S_x,\theta^u_x}$ and $R'(u,S) = [\![v]\!]^I_{\xi'^S_x,(\sigma\theta)^u_x}$. By induction hypothesis, $R(u,S) = [\![v]\!]^I_{\xi'',\sigma(\theta^u_x)}$ with $y\xi'' = [\![y\sigma]\!]_{\xi^S_x,\theta^u_x}$. Since $\sigma(\theta^u_x) = (\sigma\theta)^u_x$ and $\xi'' = \xi'^S_x$, we have $R = R'$.

- Let $R = [\![t\sigma u\sigma]\!]^I_{\xi,\theta} = [\![t\sigma]\!]^I_{\xi,\theta}(u\sigma\theta, [\![u\sigma]\!]^I_{\xi,\theta})$ and $R' = [\![tu]\!]^I_{\xi',\sigma\theta} = [\![t]\!]^I_{\xi',\sigma\theta}(u\sigma\theta, [\![u]\!]^I_{\xi',\sigma\theta})$. By induction hypothesis, $[\![t\sigma]\!]^I_{\xi,\theta} = [\![t]\!]^I_{\xi',\sigma\theta}$ and $[\![u\sigma]\!]^I_{\xi,\theta} = [\![u]\!]^I_{\xi',\sigma\theta}$. Therefore, $R = R'$.

□

6.3. *Interpretation of constant predicate symbols*

Like Mendler (Mendler 1987) or Werner (Werner 1994), we define the interpretation of constant predicate symbols as the fixpoint of some monotonic function on a complete lattice. The monotonicity is ensured by the positivity conditions of admissible inductive structures (Definition 20). The main difference with these works is that we have a more general notion of constructor since it includes any function symbol whose output type is a constant predicate symbol. This allows us to define functions and predicates by matching not only on constant constructors but also on defined symbols.

Definition 41 (Monotonic interpretation) Let I be an interpretation of $C : (\vec{x} : \vec{T})\star$, $\vec{a} = (\vec{t}, \vec{S})$[§§] and $\vec{a}' = (\vec{t}', \vec{S}')$ be arguments of I. Let $\vec{a} \leq_i \vec{a}'$ iff $\vec{t} = \vec{t}'$, $S_i \leq S'_i$ and, for all $j \neq i$, $S_j = S'_j$. We say that I is *monotonic* if, for all $i \in \mathrm{Mon}(C)$, $\vec{a} \leq_i \vec{a}' \Rightarrow I(\vec{a}) \leq I(\vec{a}')$.

We define the monotonic interpretation I of $\mathcal{CF}^\Box$ by induction on $>_\mathcal{C}$ **(A2)**. Let $C \in \mathcal{CF}^\Box$ and assume that we already defined a monotonic interpretation K for every symbol smaller than C. Let $\mathcal{I}$ (resp. $\mathcal{I}^m$) be the set of (resp. monotonic) interpretations of $\{D \in \mathcal{CF}^\Box \mid D =_\mathcal{C} C\}$, and $\leq$ be the relation on $\mathcal{I}$ defined by $I \leq I'$ iff, for all $D =_\mathcal{C} C$, $I_D \leq_{\tau_D} I'_D$. For simplicity, we denote $[\![t]\!]^{K \cup I}$ by $[\![t]\!]^I$.

Lemma 42 $(\mathcal{I}^m, \leq)$ is a complete lattice.

Proof. First of all, $\leq$ is an ordering since, for all $D =_\mathcal{C} C$, $\leq_{\tau_D}$ is an ordering.

The function $I^\top$ defined by $I^\top_D = \top_{\tau_D}$ is the greatest element of $\mathcal{I}$. We show that it belongs to $\mathcal{I}^m$. Let $D =_\mathcal{C} C$ with $D : (\vec{x} : \vec{T})U$, $i \in \mathrm{Mon}(D)$ and $\vec{a} \leq_i \vec{a}'$. Then, $I^\top_D(\vec{a}) = \top_U = I^\top_D(\vec{a}')$.

We now show that every part of $\mathcal{I}^m$ has an inf. Let $\Im \subseteq \mathcal{I}^m$ and $I^\wedge$ be the function defined by $I^\wedge_D = \bigwedge_{\tau_D}(\Re_D)$ where $\Re_D = \{I_D \mid I \in \Im\}$. We show that $I^\wedge \in \mathcal{I}^m$. Let $D =_\mathcal{C} C$ with $D : (\vec{x} : \vec{T})U$, $i \in \mathrm{Mon}(D)$ and $\vec{a} \leq_i \vec{a}'$. Then, $I^\wedge_D(\vec{a}) = \bigwedge_U\{I_D(\vec{a}) \mid I \in \Im\}$ and $I^\wedge_D(\vec{a}') = \bigwedge_U\{I_D(\vec{a}') \mid I \in \Im\}$. Since each I_D is monotonic, $I_D(\vec{a}) \leq_U I_D(\vec{a}')$. Therefore, $I^\wedge_D \leq_{\tau_D} I^\wedge_D$.

We are left to show that $I^\wedge$ is the inf of $\Im$. For all $I \in \Im$, $I^\wedge \leq I$ since, for all $D =_\mathcal{C} C$, $I^\wedge_D$ is the inf of $\Re_D$. Assume now that there exists $I' \in \mathcal{I}^m$ such that, for all $I \in \Im$, $I' \leq I$. Then $I' \leq I^\wedge$ since $I^\wedge_D$ is the inf of $\Re_D$. □

Definition 43 (Interpretation of constant predicate symbols) Let φ be the function which, to $I \in \mathcal{I}^m$, associates the interpretation $\varphi^I \in \mathcal{I}^m$ such that $\varphi^I_D(\vec{t}, \vec{S})$ is the set of terms $u \in \mathcal{SN}$ such that if u reduces to $f\vec{u}$ with $f : (\vec{y} : \vec{U})D\vec{v}$ and $|\vec{u}| = |\vec{y}|$ then,

[§§] For simplicity, we write $(\vec{t}, \vec{S})$ instead of $(t_1, S_1), \ldots, (t_n, S_n)$.

for all $j \in \mathrm{Acc}(f)$, $u_j \in [\![U_j]\!]^I_{\xi,\theta}$ with $\theta = \{\vec{y} \mapsto \vec{u}\}$ and $y\xi = S_{\iota_y}$. We show hereafter that φ is monotonic. Therefore, we can take $I = \mathrm{lfp}(\varphi)$, the least fixpoint of φ.

Since $\varphi^I_D(\vec{t}, \vec{S})$ does not depend on $\vec{t}$, we may sometimes write $I_D(\vec{S})$ instead of $I_D(\vec{t}, \vec{S})$. The aim of this definition is to ensure the correctness of the accessibility relations (Lemma 53): if $f\vec{u}$ is computable then each accessible u_j is computable. This will allow us to ensure the computability of the variables of the left hand-side of a rule if the arguments of the left hand-side are computable, and thus the computability of the right hand-sides that belong to the computability closure.

Lemma 44 φ^I is a well defined interpretation.

Proof. We first prove that φ^I is well defined. The existence of ι_y is the hypothesis **(I6)**. The interpretations necessary for computing $[\![U_j]\!]_{\xi,\theta}$ are all well defined. The interpretation of constant predicate symbols smaller than D is K. The interpretation of constant predicate symbols equivalent to D is I. By **(I4)** and **(I5)**, constant predicate symbols greater than D and defined predicate symbols do not occur in U_j. Finally, we must make sure that $\xi \models \Gamma$ where Γ is the environment made of the declarations $y_i : U_i$ such that $y_i \in \mathrm{FV}^{\square}(U_j)$ for some j. Let $y \in \mathrm{dom}(\Gamma)$. We must prove that $y\xi \in \mathcal{R}_{y\Gamma}$. Assume that $D : (\vec{x} : \vec{T})U$. Then, $y\xi = S_{\iota_y} \in \mathcal{R}_{T_{\iota_y}}$. Let $\gamma = \{\vec{x} \mapsto \vec{v}\}$. Since $\gamma : \Gamma_D \rightsquigarrow \Gamma_f$, by Lemma 34, $\mathcal{R}_{T_{\iota_y}} = \mathcal{R}_{T_{\iota_y}\gamma}$. By **(I6)**, $v_{\iota_y} = y$. So, $\Gamma_f \vdash y : T_{\iota_y}\gamma$ and $T_{\iota_y}\gamma \ \mathbb{C}^*_{\Gamma_f}\ y\Gamma$. Therefore, by Lemma 34, $\mathcal{R}_{T_{\iota_y}\gamma} = \mathcal{R}_{y\Gamma}$ and $y\xi \in \mathcal{R}_{y\Gamma}$.

We now prove that $\varphi^I_D \in \mathcal{R}_{\tau_D}$. It is clearly stable by reduction since it does not depend on $\vec{t}$. Furthermore, $R = \varphi^I_D(\vec{t}, \vec{S})$ satisfies the properties (R1) to (R3):

(R1) Strong normalization. By definition.

(R2) Stability by reduction. Let $u \in R$ and $u' \in \rightarrow(u)$. Since $u \in \mathcal{SN}$, $u' \in \mathcal{SN}$. Assume furthermore that $u' \rightarrow^* f\vec{u}$ with $f : (\vec{y} : \vec{U})D\vec{v}$. Then, $u \rightarrow^* f\vec{u}$. Therefore, for all $j \in \mathrm{Acc}(f)$, $u_j \in [\![U_j]\!]_{\xi,\theta}$ and $u' \in R$.

(R3) Neutral terms. Let u be a neutral term such that $\rightarrow(u) \subseteq R$. Then, $u \in \mathcal{SN}$. Assume now that $u \rightarrow^* f\vec{u}$ with $f : (\vec{y} : \vec{U})D\vec{v}$. Since u is neutral, $u \neq f\vec{u}$ and there exists $u' \in \rightarrow(u)$ such that $u' \rightarrow^* f\vec{u}$. Therefore, for all $j \in \mathrm{Acc}(f)$, $u_j \in [\![U_j]\!]_{\xi,\theta}$ and $u \in R$.

□

Lemma 45 Let $\leq^+ = \leq$, $\leq^- = \geq$ and $\xi \leq_x \xi'$ iff $x\xi \leq x\xi'$ and, for all $y \neq x$, $y\xi = y\xi'$. If I is monotonic, $\xi \leq_x \xi'$, $\mathrm{Pos}(x,t) \subseteq \mathrm{Pos}^\delta(t)$, $\Gamma \vdash t : T$ and $\xi, \xi' \models \Gamma$ then $[\![t]\!]^I_{\xi,\theta} \leq^\delta [\![t]\!]^I_{\xi',\theta}$.

Proof. By induction on t.

- $[\![s]\!]^I_{\xi,\theta} = \top_s = [\![s]\!]^I_{\xi',\theta}$.
- $[\![x]\!]^I_{\xi,\theta} = x\xi \leq x\xi' = [\![x]\!]^I_{\xi',\theta}$ and $\delta = +$ necessarily.
- $[\![y]\!]^I_{\xi,\theta} = y\xi = y\xi' = [\![y]\!]^I_{\xi',\theta}$ $(y \neq x)$.
- Let $R = [\![F\vec{t}]\!]^I_{\xi,\theta}$ and $R' = [\![F\vec{t}]\!]^I_{\xi',\theta}$. $R = I_F(\vec{a})$ with $a_i = (t_i\theta, [\![t_i]\!]^I_{\xi,\theta})$ and $R' = I_F(\vec{a}')$ with $a'_i = (t_i\theta, [\![t_i]\!]^I_{\xi',\theta})$. $\mathrm{Pos}^\delta(F\vec{t}) = \{1^{|\vec{t}|} \mid \delta = +\} \cup \bigcup\{1^{|\vec{t}|-i}2.\mathrm{Pos}^\delta(t_i) \mid i \in \mathrm{Mon}(F)\}$. If $i \in \mathrm{Mon}(F)$ then $\mathrm{Pos}(x,t_i) \subseteq \mathrm{Pos}^\delta(t_i)$ and, by induction hypothesis, $[\![t_i]\!]^I_{\xi,\theta} \leq^\delta [\![t_i]\!]^I_{\xi',\theta}$.

Otherwise, $\mathrm{Pos}(x,t_i)=\emptyset$ and $[\![t_i]\!]^I_{\xi,\theta}=[\![t_i]\!]^I_{\xi',\theta}$. Therefore, in both cases, $R\leq^\delta R'$ since I_F is monotonic.

- Let $R=[\![(x:U)V]\!]^I_{\xi,\theta}$ and $R'=[\![(x:U)V]\!]^I_{\xi',\theta}$. $R=\{t\in\mathcal{T}\mid\forall u\in[\![U]\!]^I_{\xi,\theta},\forall S\in\mathcal{R}_U,tu\in[\![V]\!]^I_{\xi^S_x,\theta^u_x}\}$. $R'=\{t\in\mathcal{T}\mid\forall u\in[\![U]\!]^I_{\xi',\theta},\forall S\in\mathcal{R}_U,tu\in[\![V]\!]^I_{\xi'^S_x,\theta^u_x}\}$. Since $\mathrm{Pos}^\delta((x:U)V)=1.\mathrm{Pos}^{-\delta}(U)\cup 2.\mathrm{Pos}^\delta(V)$, $\mathrm{Pos}(x,U)\subseteq\mathrm{Pos}^{-\delta}(U)$ and $\mathrm{Pos}(x,V)\subseteq\mathrm{Pos}^\delta(V)$. Therefore, by induction hypothesis, $[\![U]\!]^I_{\xi,\theta}\leq^{-\delta}[\![U]\!]^I_{\xi',\theta}$ and $[\![V]\!]^I_{\xi^S_x,\theta^u_x}\leq^\delta[\![V]\!]^I_{\xi'^S_x,\theta^u_x}$. So, $R\leq^\delta R'$. Indeed, if $\delta=+$, $t\in R$ and $u\in[\![U]\!]^I_{\xi',\theta}\subseteq[\![U]\!]^I_{\xi,\theta}$ then $tu\in[\![V]\!]^I_{\xi^S_x,\theta^u_x}\subseteq[\![V]\!]^I_{\xi'^S_x,\theta^u_x}$ and $t\in R'$. If $\delta=-$, $t\in R'$ and $u\in[\![U]\!]^I_{\xi,\theta}\subseteq[\![U]\!]^I_{\xi',\theta}$ then $tu\in[\![V]\!]^I_{\xi'^S_x,\theta^u_x}\subseteq[\![V]\!]^I_{\xi^S_x,\theta^u_x}$ and $t\in R$.
- Let $R=[\![[x:U]v]\!]^I_{\xi,\theta}$ and $R'=[\![[x:U]v]\!]^I_{\xi',\theta}$. R and R' have the same domain $\mathcal{T}\times\mathcal{R}_U$ and the same codomain $\mathcal{R}_V$. $R(u,S)=[\![v]\!]^I_{\xi^S_x,\theta^u_x}$ and $R'(u,S)=[\![v]\!]^I_{\xi'^S_x,\theta^u_x}$. Since $\mathrm{Pos}^\delta([x:U]v)=2.\mathrm{Pos}^\delta(v)$, $\mathrm{Pos}(x,v)\subseteq\mathrm{Pos}^\delta(v)$. Therefore, by induction hypothesis, $R(u,S)\leq^\delta R'(u,S)$ and $R\leq^\delta R'$.
- Let $R=[\![tu]\!]^I_{\xi,\theta}$ and $R'=[\![tu]\!]^I_{\xi',\theta}$ ($t\neq f\vec{t}$). $R=[\![t]\!]^I_{\xi,\theta}(u\theta,S)$ with $S=[\![u]\!]^I_{\xi,\theta}$. $R'=[\![t]\!]^I_{\xi',\theta}(u\theta,S')$ with $S'=[\![u]\!]^I_{\xi',\theta}$. Since $\mathrm{Pos}^\delta(tu)=1.\mathrm{Pos}^\delta(t)$, $\mathrm{Pos}(x,t)\subseteq\mathrm{Pos}^\delta(t)$ and $\mathrm{Pos}(x,u)=\emptyset$. Therefore, $S=S'$ and, by induction hypothesis, $[\![t]\!]^I_{\xi,\theta}\leq^\delta[\![t]\!]^I_{\xi',\theta}$. So, $R\leq^\delta R'$.

□

Lemma 46 φ^I is monotonic.

Proof. Let $D=_{\mathcal{C}}C$ with $D:(\vec{x}:\vec{T})U$, $i\in\mathrm{Mon}(D)$ and $\vec{a}\leq_i\vec{a}'$ with $\vec{a}=(\vec{t},\vec{S})$ and $\vec{a}'=(\vec{t},\vec{S}')$. We have to show that $\varphi^I_D(\vec{a})\subseteq\varphi^I_D(\vec{a}')$. Let $u\in\varphi^I_D(\vec{a})$. We prove that $u\in\varphi^I_D(\vec{a}')$. First, we have $u\in\mathcal{SN}$. Assume now that u reduces to $f\vec{u}$ with $f:(\vec{y}:\vec{U})D\vec{v}$. Let $j\in\mathrm{Acc}(f)$. We have to prove that $u_j\in[\![U_j]\!]_{\xi',\theta}$ with $\theta=\{\vec{y}\mapsto\vec{u}\}$ and, for all $y\in\mathrm{FV}^\square(U_j)$, $y\xi'=S'_{\iota_y}$. Since $u\in\varphi^I_D(\vec{a})$, we have $u_j\in[\![U_j]\!]_{\xi,\theta}$ with, for all $y\in\mathrm{FV}^\square(U_j)$, $y\xi=S_{\iota_y}$. If, for all $y\in\mathrm{FV}^\square(U_j)$, $\iota_y\neq i$, then ξ and ξ' are equal on $\mathrm{FV}^\square(U_j)$. Therefore, $[\![U_j]\!]_{\xi,\theta}=[\![U_j]\!]_{\xi',\theta}$ and $u_j\in[\![U_j]\!]_{\xi',\theta}$. If there exists $y\in\mathrm{FV}^\square(U_j)$ such that $\iota_y=i$ then $\xi\leq_y\xi'$. By **(I2)**, $\mathrm{Pos}(y,U_j)\subseteq\mathrm{Pos}^+(U_j)$. Therefore, by Lemma 45, $\varphi^I_D(\vec{a})\subseteq\varphi^I_D(\vec{a}')$ and $u_j\in[\![U_j]\!]_{\xi',\theta}$. □

Lemma 47 Let $I\leq_F I'$ iff $I_F\leq I'_F$ and, for all $G\neq F$, $I_G=I'_G$. If I is monotonic, $I\leq_F I'$, $\mathrm{Pos}(F,t)\subseteq\mathrm{Pos}^\delta(t)$, $\Gamma\vdash t:T$ and $\xi\models\Gamma$ then $[\![t]\!]^I_{\xi,\theta}\leq^\delta[\![t]\!]^{I'}_{\xi,\theta}$.

Proof. By induction on t.

- $[\![s]\!]^I_{\xi,\theta}=\top_s=[\![s]\!]^{I'}_{\xi,\theta}$.
- $[\![x]\!]^I_{\xi,\theta}=x\xi=[\![x]\!]^{I'}_{\xi,\theta}$.
- Let $R=[\![G\vec{t}]\!]^I_{\xi,\theta}$ and $R'=[\![G\vec{t}]\!]^{I'}_{\xi,\theta}$. $R=I_G(\vec{a})$ with $a_i=(t_i\theta,[\![t_i]\!]^I_{\xi,\theta})$. $R'=I'_G(\vec{a}')$ with $a'_i=(t_i\theta,[\![t_i]\!]^{I'}_{\xi,\theta})$. $\mathrm{Pos}^\delta(G\vec{t})=\{1^{|\vec{t}|}\mid\delta=+\}\cup\bigcup\{1^{|\vec{t}|-i}2.\mathrm{Pos}^\delta(t_i)\mid i\in\mathrm{Mon}(G)\}$. If $i\in\mathrm{Mon}(G)$ then $\mathrm{Pos}(F,t_i)\subseteq\mathrm{Pos}^\delta(t_i)$ and, by induction hypothesis, $[\![t_i]\!]^I_{\xi,\theta}\leq^\delta[\![t_i]\!]^{I'}_{\xi,\theta}$. Otherwise, $\mathrm{Pos}(F,t_i)=\emptyset$ and $[\![t_i]\!]^I_{\xi,\theta}=[\![t_i]\!]^{I'}_{\xi,\theta}$. Therefore, $I_G(\vec{a})\leq^\delta I_G(\vec{a}')$ since I_G is

monotonic. Now, if $G = F$ then $\delta = +$ and $I_G(\vec{a}) \leq I_G(\vec{a}') = I_F(\vec{a}') \leq I'_F(\vec{a}') = I'_G(\vec{a}')$. Otherwise, $I_G(\vec{a}) \leq^\delta I_G(\vec{a}') = I'_G(\vec{a}')$.

- Let $R = [\![(x : U)V]\!]^I_{\xi,\theta}$ and $R' = [\![(x : U)V]\!]^{I'}_{\xi,\theta}$. $R = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]^I_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^I_{\xi^S_x,\theta^u_x}\}$ and $R' = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]^{I'}_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]^{I'}_{\xi^S_x,\theta^u_x}\}$. Since $\mathrm{Pos}^\delta((x : U)V) = 1.\mathrm{Pos}^{-\delta}(U) \cup 2.\mathrm{Pos}^\delta(V)$, $\mathrm{Pos}(F, U) \subseteq \mathrm{Pos}^{-\delta}(U)$ and $\mathrm{Pos}(F, V) \subseteq \mathrm{Pos}^\delta(V)$. Therefore, by induction hypothesis, $[\![U]\!]^I_{\xi,\theta} \leq^{-\delta} [\![U]\!]^{I'}_{\xi,\theta}$ and $[\![V]\!]^I_{\xi^S_x,\theta^u_x} \leq^\delta [\![V]\!]^{I'}_{\xi^S_x,\theta^u_x}$. So, $[\![t]\!]^I_{\xi,\theta} \leq^\delta [\![t]\!]^{I'}_{\xi,\theta}$. Indeed, if $\delta = +$, $t \in R$ and $u \in [\![U]\!]^{I'}_{\xi,\theta} \subseteq [\![U]\!]^I_{\xi,\theta}$ then $tu \in [\![V]\!]^I_{\xi^S_x,\theta^u_x} \subseteq [\![V]\!]^{I'}_{\xi^S_x,\theta^u_x}$ and $t \in R'$. If $\delta = -$, $t \in R'$ and $u \in [\![U]\!]^I_{\xi,\theta} \subseteq [\![U]\!]^{I'}_{\xi,\theta}$ then $tu \in [\![V]\!]^{I'}_{\xi^S_x,\theta^u_x} \subseteq [\![V]\!]^I_{\xi^S_x,\theta^u_x}$ and $t \in R$.
- Let $R = [\![[x : U]v]\!]^I_{\xi,\theta}$ and $R' = [\![[x : U]v]\!]^{I'}_{\xi,\theta}$. R and R' have the same domain $\mathcal{T} \times \mathcal{R}_U$ and same codomain $\mathcal{R}_V$. $R(u, S) = [\![v]\!]^I_{\xi^S_x,\theta^u_x}$ and $R'(u, S) = [\![v]\!]^{I'}_{\xi^S_x,\theta^u_x}$. Since $\mathrm{Pos}^\delta([x : U]v) = 2.\mathrm{Pos}^\delta(v)$, $\mathrm{Pos}(F, v) \subseteq \mathrm{Pos}^\delta(v)$. Therefore, by induction hypothesis, $R(u, S) \leq^\delta R'(u, S)$ and $R \leq^\delta R'$.
- Let $R = [\![tu]\!]^I_{\xi,\theta}$ and $R' = [\![tu]\!]^{I'}_{\xi,\theta}$ $(t \neq f\vec{t})$. $R = [\![t]\!]^I_{\xi,\theta}(u\theta, S)$ with $S = [\![u]\!]^I_{\xi,\theta}$. $R' = [\![t]\!]^{I'}_{\xi,\theta}(u\theta, S')$ with $S' = [\![u]\!]^{I'}_{\xi,\theta}$. Since $\mathrm{Pos}^\delta(tu) = 1.\mathrm{Pos}^\delta(t)$, $\mathrm{Pos}(F, t) \subseteq \mathrm{Pos}^\delta(t)$ and $\mathrm{Pos}(F, u) = \emptyset$. Therefore, $S = S'$ and, by induction hypothesis, $[\![t]\!]^I_{\xi,\theta} \leq^\delta [\![t]\!]^{I'}_{\xi,\theta}$. So, $R \leq^\delta R'$.

□

Lemma 48 φ is monotonic.

Proof. Let $I, I' \in \mathcal{I}^m$ such that $I \leq I'$. We have to prove that, for all $D =_\mathcal{C} C$, $\varphi^I_D \leq \varphi^{I'}_D$, that is, $\varphi^I_D(\vec{a}) \subseteq \varphi^{I'}_D(\vec{a})$ for all $\vec{a}$. Let $u \in \varphi^I_D(\vec{a})$. We prove that $u \in \varphi^{I'}_D(\vec{a})$. First, we have $u \in \mathcal{SN}$. Assume now that u reduces to $f\vec{u}$ with $f : (\vec{y} : \vec{U})D\vec{v}$. Let $j \in \mathrm{Acc}(f)$. We have to prove that $u_j \in [\![U_j]\!]^{I'}_{\xi,\theta}$ with $\theta = \{\vec{y} \mapsto \vec{u}\}$ and, for all $y \in \mathrm{FV}^\square(U_j)$, $y\xi = S_{\iota_y}$. Since $u \in \varphi^I_D(\vec{a})$, we have $u_j \in [\![U_j]\!]^I_{\xi,\theta}$. Since $j \in \mathrm{Acc}(f)$, by **(I3)**, for all $E =_\mathcal{C} D$, $\mathrm{Pos}(E, U_j) \subseteq \mathrm{Pos}^+(U_j)$. Now, only a finite number of symbols $E =_\mathcal{C} D$ can occur in U_j, say $E_0, \ldots, E_{n-1}$. Let $I^0 = I$ and, for all $i < n$, $I^{i+1}_D = I^i_D$ if $D \neq E_i$, and $I^{i+1}_D = I'_{E_i}$ otherwise. We have $I = I^0 \leq_{E_0} I^1 \leq_{E_1} \ldots I^{n-1} \leq_{E_{n-1}} I^n = I'$. Hence, by Lemma 47, $[\![U_j]\!]^I_{\xi,\theta} \leq [\![U_j]\!]^{I'}_{\xi,\theta}$ and $u \in \varphi^{I'}_D(\vec{a})$. □

Since $(\mathcal{I}^m, \leq)$ is a complete lattice, φ has a least fixpoint I which is an interpretation for all the constant predicate symbols equivalent to C. Hence, by induction on $>_\mathcal{C}$, we obtain an interpretation I for all the constant predicate symbols.

In the case of a primitive constant predicate symbol, the interpretation is simply the set of strongly normalizable terms of this type:

Lemma 49 (Interpretation of primitive constant predicate symbols) If C is a primitive constant predicate symbol then $I_C = \top_{\tau_C}$.

Proof. Since $I_C \leq \top_{\tau_C}$, it suffices to prove that $\top_{\tau_C} \leq I_C$. Since, by assumption, $\vdash \tau_C : \square$, τ_C is of the form $(\vec{x} : \vec{T})\star$. If $\vec{a}$ are arguments of $\top_{\tau_C}$ then $\top_{\tau_C}(\vec{a}) = \top_\star = \mathcal{SN}$ and it suffices to prove that, for all $u \in \mathcal{SN}$, C primitive and $\vec{a}$ arguments of I_C, $u \in I_C(\vec{a})$,

by induction on u with $\to \cup \rhd$ as well-founded ordering. Assume that $u \to^* f\vec{u}$ with $f : (\vec{y} : \vec{U})C\vec{v}$. If $u \to^+ f\vec{u}$, we can conclude by induction hypothesis. So, assume that $u = f\vec{u}$. In this case, we have to prove that, for all $j \in \mathrm{Acc}(f)$, $u_j \in [\![U_j]\!]_{\xi,\theta}$ with $\theta = \{\vec{y} \mapsto \vec{u}\}$ and, for all $y \in \mathrm{FV}^{\square}(U_j)$, $y\xi = S_{\iota_y}$. By definition of primitive constant predicate symbols, for all $j \in \mathrm{Acc}(f)$, U_j is of the form $D\vec{w}$ with D primitive too. Hence, $[\![U_j]\!]_{\xi,\theta} = I_D(\vec{a}')$ with $a'_i = (w_i\theta, [\![w_i]\!]_{\xi,\theta})$. Since $u_j \in \mathcal{SN}$, by induction hypothesis, $u_j \in I_D(\vec{a}')$. Therefore, $u \in I_C(\vec{a})$. □

6.4. *Computability ordering*

In this section, we assume given an interpretation J for defined predicate symbols and denote $[\![T]\!]^{I\cup J}$ by $[\![T]\!]$. The fixpoint of the function φ defined in the previous section can be reached by transfinite iteration from the smallest element of $\mathcal{I}^m$, $\bot_C(\vec{t}, \vec{S}) = \bot_\star$. Let $I^{\mathfrak{a}}$ be the interpretation reached after $\mathfrak{a}$ iterations of φ.

Definition 50 (Order of a computable term) The *order* of a term $t \in I_C(\vec{S})$, written $o_{C(\vec{S})}(t)$, is the smallest ordinal $\mathfrak{a}$ such that $t \in I^{\mathfrak{a}}_C(\vec{S})$.

This notion of order will enable us to define a well-founded ordering in which recursive definitions on strictly positive predicates strictly decrease. Indeed, in this case, the subterm ordering is not sufficient. In the example of the addition on ordinals, we have the rule:

$$x + (lim\ f) \quad \to \quad lim\ ([n : nat]x + fn)$$

We have a recursive call with (fn) as argument, which is not a subterm of $(lim\ f)$. However, thanks to the definition of the interpretation for constant predicate symbols and products, we can say that, if $(lim\ f)$ is computable then f is computable and thus that, for all computable n, (fn) is computable. So, the order of $(lim\ f)$ is greater than the one of (fn): $o(lim\ f) > o(fn)$.

Definition 51 (Computability ordering) Let $f \in \mathcal{F}$ with $stat_f = lex\ m_1 \dots m_k$. Let Θ_f be the set of tuples (g, ξ, θ) such that $g =_{\mathcal{F}} f$ and $\xi, \theta \models \Gamma_g$. We equip Θ_f with the ordering $\sqsupset_f$ defined by:

- $(g, \xi, \theta) \sqsupset_f (g', \xi', \theta')$ if $\vec{m}\theta\ (\sqsupset_f^{1,m}, \dots, \sqsupset_f^{k,m})_{\mathrm{lex}}\ \vec{m}\theta'$,
- $mul\ \vec{t} \sqsupset_f^{i,m} mul\ \vec{t}'$ if $\{\vec{t}\}\ (\sqsupset_f^i)_{\mathrm{mul}}\ \{\vec{t}'\}$,
- $t \sqsupset_f^i t'$ if $i \in SP(f)$, $T_f^i = C\vec{a}$, $[\![\vec{a}]\!]_{\xi,\theta} = [\![\vec{a}]\!]_{\xi',\theta'} = \vec{S}$ and $o_{C(\vec{S})}(t) > o_{C(\vec{S})}(t')$,
- $t \sqsupset_f^i t'$ if $i \notin SP(f)$ and $t\,(\to \cup \rhd)\,t'$.

We equip $\Theta = \bigcup\{\Theta_f \mid f \in \mathcal{F}\}$ with the *computability ordering* $\sqsupset$ defined by $(f, \xi, \theta) \sqsupset (f', \xi', \theta')$ if $f >_{\mathcal{F}} f'$ or, $f =_{\mathcal{F}} f'$ and $(f, \xi, \theta) \sqsupset_f (f', \xi', \theta')$.

Lemma 52 The computability ordering is well-founded and compatible with $\to$, that is, if $\theta \to \theta'$ then $(g, \xi, \theta) \sqsupseteq (g, \xi, \theta')$.

Proof. The computability ordering is well-founded since ordinals are well-founded and lexicographic and multiset orderings preserve well-foundedness. It is compatible with $\rightarrow$ by definition of the interpretation of constant predicate symbols. □

We check hereafter that the accessibility relation is correct, that is, an accessible subterm of a computable term is computable. Then, we check that the ordering on arguments is correct too, that is, if $t >_R^i u$ and t is computable then u is computable and $o(t) > o(u)$.

Lemma 53 (Correctness of accessibility) If $t : T \rhd_\rho u : U$ and $t\sigma \in [\![T\rho]\!]^{I^{\mathfrak{a}}}_{\xi,\sigma}$ with $\mathfrak{a}$ as small as posssible then $\mathfrak{a} = \mathfrak{b} + 1$ and $u\sigma \in [\![U\rho]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$.

Proof. By definition of $\rhd_\rho$, we have $t = f\vec{u}$, $f : (\vec{y} : \vec{U})C\vec{v}$, $C \in \mathcal{CF}^{\Box}$, $u = u_j$, $j \in \mathrm{Acc}(f)$, $T\rho = C\vec{v}\gamma\rho$, $U\rho = U_j\gamma\rho$, $\gamma = \{\vec{y} \mapsto \vec{u}\}$ and no $D =_{\mathcal{C}} C$ occurs in $\vec{u}\rho$. Hence, $t\sigma \in [\![C\vec{v}\gamma\rho]\!]^{I^{\mathfrak{a}}}_{\xi,\sigma} = I^{\mathfrak{a}}_C(\vec{S})$ with $\vec{S} = [\![\vec{v}\gamma\rho]\!]^{I^{\mathfrak{a}}}_{\xi,\sigma}$. Assume that $\mathfrak{a} = 0$. Then, $I^{\mathfrak{a}}_C(\vec{S}) = \bot_\star$. But $f\vec{u} \notin \bot_\star$ since $f\vec{u}$ is not neutral (see Lemma 36). So, $\mathfrak{a} \neq 0$. Assume now that $\mathfrak{a}$ is a limit ordinal. Then, $I^{\mathfrak{a}}_C(\vec{S}) = \bigcup\{I^{\mathfrak{b}}_C(\vec{S}) \mid \mathfrak{b} < \mathfrak{a}\}$ and $t\sigma \in I^{\mathfrak{b}}_C(\vec{S})$ for some $\mathfrak{b} < \mathfrak{a}$, which is not possible since $\mathfrak{a}$ is as small as possible. Therefore, $\mathfrak{a} = \mathfrak{b} + 1$ and, by definition of I_C, $u_j\sigma \in [\![U_j]\!]^{I^{\mathfrak{b}}}_{\xi',\gamma\rho\sigma}$ with $y\xi' = S_{\iota_y}$. By **(I6)**, $v_{\iota_y} = y$. Thus, $y\xi' = [\![y\gamma\rho]\!]^{I^{\mathfrak{a}}}_{\xi,\sigma}$. Now, since no $D =_{\mathcal{C}} C$ occurs in $\vec{u}\rho$, $y\xi' = [\![y\gamma\rho]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$. Hence, by candidate substitution, $[\![U_j]\!]^{I^{\mathfrak{b}}}_{\xi',\gamma\rho\sigma} = [\![U_j\gamma\rho]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$ and $u\sigma \in [\![U\rho]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$ since $U\rho = U_j\gamma\rho$. □

Lemma 54 (Correctness of the ordering on arguments) Assume that $t : T >_R^i u : U$ as in Definition 24, $t\sigma \in [\![T\rho]\!]_{\xi,\sigma}$ and $\vec{u}\sigma \in [\![\vec{U}\delta]\!]_{\xi,\sigma}$. Then, $u\sigma \in [\![U\rho]\!]_{\xi,\sigma}$ and $o_{C(\vec{S})}(t\sigma) > o_{C(\vec{S})}(u\sigma)$ with $\vec{S} = [\![\vec{v}\gamma\rho]\!]_{\xi,\sigma}$.

Proof. Since $t : T \rhd^+_\rho x : V$, $T\rho = C\vec{v}\gamma\rho$. Hence, $t\sigma \in I^{\mathfrak{a}}_C(\vec{S})$ with $\mathfrak{a} = o_{C(\vec{S})}(t\sigma)$. By Lemma 53, $\mathfrak{a} = \mathfrak{b}+1$ and $x\sigma \in [\![V\rho]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$. Since no $D =_{\mathcal{C}} C$ occurs in $\vec{U}\delta$, $[\![\vec{U}\delta]\!]_{\xi,\sigma} = [\![\vec{U}\delta]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$. Since $V\rho = (\vec{y} : \vec{U})C\vec{w}$ and $\vec{u}\sigma \in [\![\vec{U}\delta]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$, $u\sigma \in [\![C\vec{w}]\!]^{I^{\mathfrak{b}}}_{\xi^{\vec{R}}_{\vec{y}},\sigma^{\vec{u}\sigma}_{\vec{y}}}$ with $\vec{R} = [\![\vec{u}]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$. By candidate substitution, $[\![C\vec{w}]\!]^{I^{\mathfrak{b}}}_{\xi^{\vec{R}}_{\vec{y}},\sigma^{\vec{u}\sigma}_{\vec{y}}} = [\![C\vec{w}\delta]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma} = I^{\mathfrak{b}}_C(\vec{S}')$ with $\vec{S}' = [\![\vec{w}\delta]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$. Since $\vec{w}\delta|_C = \vec{v}\gamma\rho|_C$, $\vec{S}' = [\![\vec{v}\gamma\rho]\!]^{I^{\mathfrak{b}}}_{\xi,\sigma}$. Since no $D =_{\mathcal{C}} C$ occurs in $\vec{v}\gamma\rho$, $\vec{S}' = \vec{S}$. Therefore, $u\sigma \in I^{\mathfrak{b}}_C(\vec{S})$ and $o_{C(\vec{S})}(t\sigma) > o_{C(\vec{S})}(u\sigma)$. □

6.5. *Interpretation of defined predicate symbols*

We define the interpretation J for defined predicate symbols by induction on $\succ$ (**A3**). Let F be a defined predicate symbol and assume that we already defined an interpretation K for every symbol smaller than F. There are three cases depending on the fact that the equivalence class of F is primitive, positive or computable. For simplicity, we denote $[\![T]\!]^{I\cup K\cup J}$ by $[\![T]\!]^J$.

6.5.1. *Primitive systems*

Definition 55 For every $G \simeq F$, we take $J_G = \top_{\tau_G}$.

6.5.2. *Positive, small and simple systems* Let $\mathcal{J}$ be the set of interpretations of the symbols equivalent to F and $\leq$ be the relation on $\mathcal{J}$ defined by $J \leq J'$ if, for all $G \simeq F$, $J_G \leq_{\tau_G} J'_G$. Since $(\mathcal{R}_{\tau_G}, \leq_{\tau_G})$ is a complete lattice, it is easy to see that $(\mathcal{J}, \leq)$ is a complete lattice too.

Definition 56 Let ψ be the function which, to $J \in \mathcal{J}$ and $G \simeq F$ with $G : (\vec{x} : \vec{T})U$, associates the interpretation ψ^J_G defined by:

$$\psi^J_G(\vec{t}, \vec{S}) = \begin{cases} [\![r]\!]^J_{\xi,\sigma} & \text{if } \vec{t} \in \mathcal{WN} \cap \mathcal{CR},\ \vec{t}{\downarrow} = \vec{l}\sigma \text{ and } (G\vec{l} \to r, \Gamma, \rho) \in \mathcal{R} \\ \top_U & \text{otherwise} \end{cases}$$

where $x\xi = S_{\kappa_x}$. We show hereafter that ψ is monotonic. So, we can take $J = \mathrm{lfp}(\psi)$.

Lemma 57 ψ^J is a well defined interpretation.

Proof. By simplicity, at most one rule can be applied at the top of $G(\vec{t}{\downarrow})$. The existence of κ_x is the smallness condition **(q)**. We now prove that $\psi^J_G \in \mathcal{R}_{\tau_G}$. By **(S3)**, $\Gamma \vdash r : U\gamma\rho$ with $\gamma = \{\vec{x} \mapsto \vec{l}\}$. Now, we prove that $\xi \models \Gamma$. Let $x \in \mathrm{FV}^{\square}(r)$, $x\xi = S_{\kappa_x} \in \mathcal{R}_{x\sigma}$ since $S_{\kappa_x} \in \mathcal{R}_{t_{\kappa_x}}$ and, by smallness, $t_{\kappa_x} = l_{\kappa_x}\sigma = x\sigma$. Therefore, by Lemma 38, $[\![r]\!]_{\xi,\sigma} \in \mathcal{R}_{U\gamma\rho} = \mathcal{R}_U$. We are left to check that ψ^J_G is stable by reduction. Assume that $\vec{t} \to \vec{t'}$. By **(A1)**, $\to$ is confluent. Therefore, $\{\vec{t}\} \subseteq \mathcal{WN}$ iff $\{\vec{t'}\} \subseteq \mathcal{WN}$. Furthermore, if $\{\vec{t}\} \subseteq \mathcal{WN}$, then $\vec{t}{\downarrow} = \vec{t'}{\downarrow}$ and $\psi^J_G(\vec{t}, \vec{S}) = \psi^J_G(\vec{t'}, \vec{S})$. □

Lemma 58 ψ is monotonic.

Proof. As in Lemma 48. □

6.5.3. *Computable, small and simple systems* Let $\mathcal{D}$ be the set of tuples $(G, \vec{t}, \vec{S})$ such that $G \simeq F$, and $\{\vec{x} \mapsto \vec{S}\}, \{\vec{x} \mapsto \vec{t}\} \models \Gamma_G$. We equip $\mathcal{D}$ with the well-founded ordering $(G, \vec{t}, \vec{S}) \sqsupset_{\mathcal{D}} (G', \vec{t'}, \vec{S'})$ iff $(G, \{\vec{x} \mapsto \vec{S}\}, \{\vec{x} \mapsto \vec{t}\}) \sqsupset (G', \{\vec{x} \mapsto \vec{S'}\}, \{\vec{x} \mapsto \vec{t'}\})$ (see Definition 51).

Definition 59 We first define J' on $\mathcal{D}$ by induction on $\sqsupset_{\mathcal{D}}$. Let $G \simeq F$ with $G : (\vec{x} : \vec{T})U$.

$$J'_G(\vec{t}, \vec{S}) = \begin{cases} [\![r]\!]^{J'}_{\xi,\sigma} & \text{if } \vec{t} \in \mathcal{WN} \cap \mathcal{CR},\ \vec{t}{\downarrow} = \vec{l}\sigma \text{ and } (G\vec{l} \to r, \Gamma, \rho) \in \mathcal{R} \\ \top_U & \text{otherwise} \end{cases}$$

where $x\xi = S_{\kappa_x}$. Then, $J_G(\vec{t}, \vec{S}) = J'_G(\vec{t}{\downarrow}, \vec{S})$ if $\vec{t} \in \mathcal{WN} \cap \mathcal{CR}$, and $J_G(\vec{t}, \vec{S}) = \top_U$ otherwise.

Lemma 60 J is a well defined interpretation.

Proof. As in Lemma 57. The well-foundedness of the definition comes from Lemma 68 and Theorem 67. In Lemma 68, we show that, starting from a sequence in $\mathcal{D}$, we can apply Theorem 67 where we show that, in a recursive call $G'\vec{t'}$, $(G, \vec{t}, \vec{S}) \sqsupset (G', \vec{t'}, \vec{S'})$ for some $\vec{S'}$. □

6.6. *Correctness of the conditions*

Definition 61 (Cap and aliens) Let ζ be an injection from classes of terms modulo $\leftrightarrow^*$ to $\mathcal{X}$. The *cap* of a term t w.r.t. a set $\mathcal{G}$ of symbols is the term $cap_{\mathcal{G}}(t) = t[x_1]_{p_1} \dots [x_n]_{p_n}$ such that, for all i, $x_i = \zeta(t|_{p_i})$ and $t|_{p_i}$ is not of the form $g\vec{t}$ with $g \in \mathcal{G}$. The $t|_{p_i}$'s are the *aliens* of t. We denote by $aliens_{\mathcal{G}}(t)$ their multiset.

Lemma 62 (Pre-computability of first-order symbols) If $f \in \mathcal{F}_1$ and $\vec{t} \in \mathcal{SN}$ then $f\vec{t} \in \mathcal{SN}$.

Proof. We prove that every reduct t' of $t = f\vec{t}$ is in $\mathcal{SN}$. Hereafter, $cap = cap_{\mathcal{F}_1}$.

Case $\mathcal{R}_\omega \neq \emptyset$. By induction on $(aliens(t), cap(t))_{\text{lex}}$ with $((\rightarrow \cup \rhd)_{\text{mul}}, \rightarrow_{\mathcal{R}_1})_{\text{lex}}$ as well-founded ordering (the aliens are strongly normalizable and, by **(f)**, $\rightarrow_{\mathcal{R}_1}$ is strongly normalizing on first-order algebraic terms).

If the reduction takes place in $cap(t)$ then this is a $\mathcal{R}_1$-reduction. By **(c)**, no symbol of $\mathcal{F}_\omega$ occurs in the rules of $\mathcal{R}_1$. And, by **(d)**, the right hand-sides of the rules of $\mathcal{R}_1$ are algebraic. Therefore, $cap(t) \rightarrow_{\mathcal{R}_1} cap(t')$. By **(e)**, the rules of $\mathcal{R}_1$ are non duplicating. Therefore, $aliens(t) \rhd_{\text{mul}} aliens(t')$ and we can conclude by induction hypothesis.

If the reduction takes place in an alien then $aliens(t)$ $(\rightarrow \cup \rhd)_{\text{mul}}$ $aliens(t')$ and we can conclude by induction hypothesis.

Case $\mathcal{R}_\omega = \emptyset$. Since the t_i's are strongly normalizable and no β-reduction can take place at the top of t, t has a β-normal form. Let $cap\beta(t)$ be the cap of its β-normal form. We prove that every immediate reduct t' of t is strongly normalizable, by induction on $(\beta cap(t), aliens(t))_{\text{lex}}$ with $(\rightarrow_{\mathcal{R}_1}, (\rightarrow \cup \rhd)_{\text{mul}})_{\text{lex}}$ as well-founded ordering (the aliens are strongly normalizable and, by **(f)**, $\rightarrow_{\mathcal{R}_1}$ is strongly normalizing on first-order algebraic terms).

If the reduction takes place in $cap(t)$ then this is a $\mathcal{R}_1$-reduction. By **(d)**, the right hand-sides of the rules of $\mathcal{R}_1$ are algebraic. Therefore, t' has a β-normal form and $cap\beta(t) \rightarrow_{\mathcal{R}_1} cap\beta(t')$. Hence, we can conclude by induction hypothesis. If the reduction is a β-reduction in an alien then $cap\beta(t) = cap\beta(t')$ and $aliens(t)$ $(\rightarrow \cup \rhd)_{\text{mul}}$ $aliens(t')$. Hence, we can conclude by induction hypothesis.

We are left with the case where the reduction is a $\mathcal{R}_1$-reduction taking place in an alien u. Then, $aliens(t) \rightarrow_{\text{mul}} aliens(t')$, $cap\beta(t) \rightarrow^*_{\mathcal{R}_1} cap\beta(t')$ and we can conclude by induction hypothesis. To see that $cap\beta(t) \rightarrow^*_{\mathcal{R}_1} cap\beta(t')$, it suffices to remark that, if we β-normalize u, then all the residuals of the $\mathcal{R}_1$-redex are still reducible (left and right hand-sides of first-order rules are algebraic). □

Lemma 63 (Computability of first-order symbols) For all $f \in \mathcal{F}_1$, $f \in [\![\tau_f]\!]$.

Proof. Assume that $f : (\vec{x} : \vec{T})U$. $f \in [\![\tau_f]\!]$ iff, for all Γ_f-valid pair (ξ, θ), $f\vec{x}\theta \in R = [\![U]\!]_{\xi,\theta}$. For first-order symbols, $U = \star$ or $U = C\vec{v}$ with C primitive. If $U = \star$ then $R = \top_\star = \mathcal{SN}$. If $U = C\vec{v}$ with $C : (\vec{y} : \vec{U})V$ then $R = I_C(\vec{a})$ with $a_i = (v_i\theta, [\![v_i]\!]_{\xi,\theta})$. Since C is primitive, by Lemma 49, $I_C = \top_{\tau_C}$ and $R = \top_V$. By assumption, $\vdash \tau_C : \Box$ and $\vdash \tau_f : s_f$. After Lemma 11, $s_f = \star$ and $V = \star$. Therefore, $R = \top_\star = \mathcal{SN}$. Now, since $\xi, \theta \models \Gamma_f$, we have $x_i\theta \in [\![T_i]\!]_{\xi,\theta} \subseteq \mathcal{SN}$ by **(R1)**. Hence, by pre-computability of first-order symbols, $f\vec{x}\theta \in [\![U]\!]_{\xi,\theta}$. □

Theorem 64 (Strong normalization of $\to_{\mathcal{R}}$) The relation $\to_{\mathcal{R}}=\to_{\mathcal{R}_1} \cup \to_{\mathcal{R}_\omega}$ is strongly normalizing.

Proof. By induction on the structure of terms. The only difficult case is $f\vec{t}$. If f is first-order, we use the Lemma of pre-computability of first-order symbols. If f is higher-order, we have to show that, if $\vec{t} \in \mathcal{SN}_{\mathcal{R}}$, then $t = f\vec{t} \in \mathcal{SN}_{\mathcal{R}}$, where $\mathcal{SN}_{\mathcal{R}}$ is the set of terms that are strong normalizable w.r.t. $\to_{\mathcal{R}}$.

Let $\varpi(t) = 0$ if t is not of the form $g\vec{u}$ and $\varpi(t) = 1$ otherwise. We prove that every reduct t' of t is strongly normalizable by induction on $(f, \varpi(\vec{t}), \vec{t}, \vec{t})$ with $(>_{\mathcal{F}}, (>_{\mathbb{N}})_{stat_f}, (\rhd \cup \to_{\mathcal{R}})_{stat_f}, (\to_{\mathcal{R}})_{\text{lex}})_{\text{lex}}$ as well-founded ordering. Assume that $t' = f\vec{t'}$ with $t_i \to_{\mathcal{R}} t'_i$ and, for all $j \neq i$, $t_j = t'_j$. Then, $\vec{t}\ (\to_{\mathcal{R}})_{\text{lex}}\ \vec{t'}$ and $\varpi(t_i) \geq \varpi(t'_i)$ since if t_i is not of the form $g\vec{u}$ then t'_i is not of the form $g\vec{u}$ either.

Assume now that there exists $f\vec{l} \to r \in \mathcal{R}_\omega$ such that $\vec{t} = \vec{l}\sigma$ and $t' = r\sigma$. By **(a)**, r belongs to the computability closure of l. It is then easy to prove that $r\sigma$ is strongly normalizable by induction on the structure of r. Again, the only difficult case is $g\vec{u}$. But then, either g is smaller than f, or g is equivalent to f and its arguments are smaller than $\vec{l}$. If $l_i >_1 u_j$ then $l_i \rhd u_j$ and $\text{FV}(u_j) \subseteq \text{FV}(l_i)$. Therefore $l_i\sigma \rhd u_j\sigma$ and $\varpi(l_i\sigma) = 1 \geq \varpi(u_j\sigma)$. If now $l_i >_2 u_j$ then u_j is of the form $x\vec{v}$ and $\varpi(l_i\sigma) = 1 > \varpi(u_j\sigma) = 0$. □

Lemma 65 (Invariance by reduction) If $\Gamma \vdash t : T$, $t \to t'$, $\xi \models \Gamma$ and $t\theta \in \mathcal{WN}$ then $[\![t]\!]_{\xi,\theta} = [\![t']\!]_{\xi,\theta}$.

Proof. By induction on t. If t is an object then t' is an object too and $[\![t]\!]_{\xi,\theta} = \emptyset = [\![t']\!]_{\xi,\theta}$. Otherwise, we proceed by case on t and t':

- Let $R = [\![F\vec{l}\sigma]\!]_{\xi,\theta}$ and $R' = [\![r\sigma]\!]_{\xi,\theta}$ with $(F\vec{l} \to r, \Gamma_0, \rho) \in \mathcal{R}$. $R = I_F(\vec{a})$ with $a_i = (l_i\sigma\theta, [\![l_i\sigma]\!]_{\xi,\theta})$. By **(A3)**, there are two sub-cases:
  - F **belongs to a primitive system**. Then, $I_F = \top_{\tau_F}$ and r is of the form $[\vec{x} : \vec{T}]\ G\vec{u}$ with $G \simeq F$ or G a primitive constant predicate symbol. In both cases, $I_G = \top_{\tau_G}$. Therefore, $R = R'$.
  - F **belongs to a positive or computable, small and simple system**. Since $l_i\sigma\theta \in \mathcal{WN}$, by **(A1)**, $l_i\sigma\theta$ has a unique normal form t_i. By simplicity, the symbols in $\vec{l}$ are constant. Therefore, t_i is of the form $l_i\theta'$ with $\sigma\theta \to^* \theta'$, and $R = [\![r]\!]_{\xi',\theta'}$ with $x\xi' = [\![l_{\kappa_x}\sigma]\!]_{\xi,\theta}$. By smallness, $l_{\kappa_x} = x$ and $x\xi' = [\![x\sigma]\!]_{\xi,\theta}$. By Lemma 38, $[\![r]\!]_{\xi',\theta'} = [\![r]\!]_{\xi',\sigma\theta}$. By **(S4)**, $\sigma : \Gamma_0 \rightsquigarrow \Gamma$. Therefore, by candidate substitution, $R = R'$.
- Let $R = [\![[x : U]v\ u]\!]_{\xi,\theta}$ and $R' = [\![v\{x \mapsto u\}]\!]_{\xi,\theta}$. Let $S = [\![u]\!]_{\xi,\theta}$. $R = [\![[x : U]v]\!](u\theta, S) = [\![v]\!]_{\xi^S_x,\theta'}$ with $\theta' = \theta^{u\theta}_x = \{x \mapsto u\}\theta$. Since $\{x \mapsto u\} : (\Gamma, x : U) \to \Gamma$, by candidate substitution, $R' = [\![v]\!]_{\xi^S_x,\theta'} = R$.
- Let $R = [\![tu]\!]_{\xi,\theta}$ and $R' = [\![t'u']\!]_{\xi,\theta}$ with $t \to t'$ and $u \to u'$. $R = [\![t]\!]_{\xi,\theta}([\![u]\!]_{\xi,\theta})$ and $R' = [\![t']\!]_{\xi,\theta}(u'\theta, [\![u']\!]_{\xi,\theta})$. By induction hypothesis, $[\![t]\!]_{\xi,\theta} = [\![t']\!]_{\xi,\theta}$ and $[\![u]\!]_{\xi,\theta} = [\![u']\!]_{\xi,\theta}$. Finally, since candidates are stable by reduction, $R = R'$.
- Let $R = [\![[x : U]v]\!]_{\xi,\theta}$ and $R' = [\![[x : U']v']\!]_{\xi,\theta}$ with $U \to U'$ and $v \to v'$. Since $\mathcal{R}_U = \mathcal{R}_{U'}$, R and R' have the same domain $\mathcal{T} \times \mathcal{R}_U$ and codomain $\mathcal{R}_V$, where V is the type of v. $R(u, S) = [\![v]\!]_{\xi^S_x,\theta^u_x}$ and $R'(u, S) = [\![v']\!]_{\xi^S_x,\theta^u_x}$. By induction hypothesis, $R(u, S) = R'(u, S)$. Therefore, $R = R'$.

- Let $R = [\![(x : U)V]\!]_{\xi,\theta}$ and $R' = [\![(x : U')V']\!]_{\xi,\theta}$. $R = \{t \in \mathcal{T} \mid \forall u \in [\![U]\!]_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V]\!]_{\xi^S_x,\theta^u_x}\}$ and $R' = \{t \in \mathcal{T} \mid \forall u \in [\![U']\!]_{\xi,\theta}, \forall S \in \mathcal{R}_U, tu \in [\![V']\!]_{\xi^S_x,\theta^u_x}\}$. By induction hypothesis, $[\![U]\!]_{\xi,\theta} = [\![U']\!]_{\xi,\theta}$ and $[\![V]\!]_{\xi^S_x,\theta^u_x} = [\![V']\!]_{\xi^S_x,\theta^u_x}$. Therefore, $R = R'$.

□

Lemma 66 (Pre-computability of well-typed terms) Assume that, for all f, $f \in [\![\tau_f]\!]$. If $\Gamma \vdash t : T$ and $\xi, \theta \models \Gamma$ then $t\theta \in [\![T]\!]_{\xi,\theta}$.

Proof. By induction on $\Gamma \vdash t : T$.

(ax) $\star\theta = \star \in [\![\Box]\!]_{\xi,\theta} = \top_\Box = \mathcal{SN}$.

(symb) By assumption.

(var) $x\theta \in [\![T]\!]_{\xi,\theta}$ since θ is adapted to ξ.

(weak) By induction hypothesis.

(prod) We have to prove that $(x : U\theta)V\theta \in [\![s']\!]_{\xi,\theta} = \top_{s'} = \mathcal{SN}$. By induction hypothesis, $U\theta \in [\![s]\!]_{\xi,\theta} = \mathcal{SN}$. Now, let $\xi' = \xi^{\top_U}_x$. Since $\xi', \theta \models \Gamma, x : U$, by induction hypothesis, $V\theta \in [\![s']\!]_{\xi',\theta} = \mathcal{SN}$.

(abs) Let $t = [x : U]v$. We have to prove that $t\theta \in [\![(x : U)V]\!]_{\xi,\theta}$. First note that $U\theta, v\theta \in \mathcal{SN}$. Indeed, let $\xi' = \xi^{\top_U}_x$. Since $\xi', \theta \models \Gamma, x : U$, by induction hypothesis, $v\theta \in [\![V]\!]_{\xi',\theta}$. Furthermore, by inversion, $\Gamma \vdash U : s$ for some s. So, by induction hypothesis, $U\theta \in [\![s]\!]_{\xi,\theta} = \mathcal{SN}$. Now, let $u \in [\![U]\!]_{\xi,\theta} \subseteq \mathcal{SN}$ and $S \in \mathcal{R}_U$. We must prove that $t\theta u \in S' = [\![V]\!]_{\xi^S_x,\theta^u_x}$. Since $t\theta u$ is neutral, it suffices to prove that $\rightarrow(t\theta u) \subseteq S'$. We prove it by induction on $(U\theta, v\theta, u)$ with $\rightarrow_{\text{lex}}$ as well-founded ordering. We have $t\theta u \rightarrow v\theta\{x \mapsto u\} = v\theta'$. Since $\xi^S_x, \theta^u_x \models \Gamma, x : U$, by induction hypothesis, $v\theta' \in S'$. For the other cases, we can conclude by induction hypothesis on $(U\theta, v\theta, u)$.

(app) We have to prove that $t\theta u\theta \in [\![V\{x \mapsto u\}]\!]_{\xi,\theta}$. By induction hypothesis, $t\theta \in [\![(x : U)V]\!]_{\xi,\theta}$ and $u\theta \in [\![U]\!]_{\xi,\theta}$. Since $S = [\![u\theta]\!]_{\xi,\theta} \in \mathcal{R}_{U\theta} = \mathcal{R}_U$, by definition of $[\![(x : U)V]\!]_{\xi,\theta}$, $t\theta u\theta \in [\![V]\!]_{\xi^S_x,\theta'}$ with $\theta' = \theta^{u\theta}_x$. By candidate substitution, $[\![V\{x \mapsto u\}]\!]_{\xi,\theta} = [\![V]\!]_{\xi',\{x\mapsto u\}\theta}$ with $y\xi' = [\![y\{x \mapsto u\}]\!]_{\xi,\theta}$. Since $\xi' = \xi^S_x$ and $\{x \mapsto u\}\theta = \theta'$, $t\theta u\theta \in [\![V\{x \mapsto u\}]\!]_{\xi,\theta}$.

(conv) In (Blanqui 2001), we show that adding the hypothesis $\Gamma \vdash T : s$ does not change the typing relation. Therefore, by induction hypothesis, $t\theta \in [\![T]\!]_{\xi,\theta}$, $T\theta \in [\![s]\!]_{\xi,\theta} = \top_s = \mathcal{SN}$ and $T'\theta \in [\![s]\!]_{\xi,\theta} = \mathcal{SN}$. Hence, by invariance by reduction, $[\![T]\!]_{\xi,\theta} = [\![T']\!]_{\xi,\theta}$ and $t\theta \in [\![T']\!]_{\xi,\theta}$.

□

Theorem 67 (Computability closure correctness) Let $(f\vec{l} \rightarrow r, \Gamma, \rho)$ be a well-formed rule with $f \in \mathcal{F}_\omega$, $f : (\vec{x} : \vec{T})U$ and $\gamma = \{\vec{x} \mapsto \vec{l}\}$. Assume that $\eta, \gamma\sigma \models \Gamma_f$, $\xi, \sigma \models \Gamma$, $x\eta = [\![x\gamma\rho]\!]_{\xi,\sigma}$, and $\vec{l}\sigma \in [\![\vec{T}\gamma\rho]\!]_{\xi,\sigma}$. Assume also that:

- $\forall g <_{\mathcal{F}} f$, $g \in [\![\tau_g]\!]$,
- $\forall g =_{\mathcal{F}} f$, if $g : (\vec{y} : \vec{U})V$ and $(f, \eta, \gamma\sigma) \sqsupseteq (g, \xi'', \theta)$ then $g\vec{y}\theta \in [\![V]\!]_{\xi'',\theta}$.

If $\Delta \vdash_{\text{c}} t : T$ and $\xi\xi', \sigma\sigma' \models \Gamma, \Delta$ then $t\sigma\sigma' \in [\![T]\!]_{\xi\xi',\sigma\sigma'}$.

Proof. By induction on $\Delta \vdash_{\text{c}} t : T$, we prove that $t\sigma\sigma' \in [\![T]\!]_{\xi\xi',\sigma\sigma'}$ as in the previous lemma. We only detail the case (symb$^=$). Let $\vec{u} = \vec{y}\delta$. By induction hypothesis, $\vec{u}\sigma\sigma' \in$

$[\![\vec{U}\delta]\!]_{\xi\xi',\sigma\sigma'}$. By candidate substitution, there exists ξ'' such that $[\![\vec{U}\delta]\!]_{\xi\xi',\sigma\sigma'} = [\![\vec{U}]\!]_{\xi'',\delta\sigma\sigma'}$, $[\![V\delta]\!]_{\xi\xi',\sigma\sigma'} = [\![V]\!]_{\xi'',\delta\sigma\sigma'}$ and $\xi'' \models \Gamma_g$. Therefore, $\xi'', \delta\sigma\sigma' \models \Gamma_g$.

We now prove that $(f,\eta,\gamma\sigma) \sqsupset (g,\xi'',\delta\sigma\sigma')$. If $l_i : T_i\gamma \rhd_\rho^+ u_j : U_j\delta$. Then, $l_i \rhd u_j$ and $\mathrm{FV}(u_j) \subseteq \mathrm{FV}(l_i)$. Therefore, $l_i\sigma = l_i\sigma\sigma' \rhd u_j\sigma\sigma'$. Assume now that $l_i : T_i\gamma >_R^k u_j : U_j\delta$, $k \in SP(f)$ and $T_f^k = C\vec{a}$. By definition of $>_R^k$, $l_i = h\vec{t}'$, $h : (\vec{x}' : \vec{T}')C\vec{v}$, $u_j = x\vec{u}'$, $x \in \mathrm{dom}(\Gamma)$, $l_i : T_i\gamma \rhd_\rho^+ x : V$ and $V\rho = x\Gamma = (\vec{y}' : \vec{U}')C\vec{w}$, where $\gamma' = \{\vec{x}' \mapsto \vec{t}'\}$ and $\delta' = \{\vec{y}' \mapsto \vec{u}'\}$. We must prove that $[\![\vec{a}]\!]_{\eta,\gamma\sigma} = [\![\vec{a}]\!]_{\xi'',\delta\sigma\sigma'} = \vec{S}$ and $o_{C(\vec{S})}(l_i\sigma) > o_{C(\vec{S})}(u_j\sigma\sigma')$.

Assume that $T_i = C\vec{t}$ and $U_j = C\vec{u}$. Since $k \in SP(f)$, $\vec{t}|_C = \vec{u}|_C = \vec{a}|_C$. By definition of $\rhd_\rho$, $T_i\gamma\rho = C\vec{v}\gamma'\rho$. Hence, $\vec{a}\gamma\rho|_C = \vec{v}\gamma'\rho|_C$. By definition of $>_R^k$, $\vec{v}\gamma'\rho|_C = \vec{w}\delta'|_C$ and $U_j\delta\rho = C\vec{w}\delta'$. Therefore, $\vec{a}\gamma\rho|_C = \vec{w}\delta'|_C = \vec{u}\delta\rho|_C = \vec{a}\delta\rho|_C = \vec{a}\delta|_C$ since $\mathrm{dom}(\rho) \subseteq \mathrm{FV}(l)$, $\mathrm{FV}(\delta) \subseteq \mathrm{dom}(\Delta)$ and $\mathrm{dom}(\Delta) \cap \mathrm{FV}(l) = \emptyset$. By **(S5)**, $[\![\vec{a}]\!]_{\eta,\gamma\sigma} = [\![\vec{a}]\!]_{\eta,\gamma\rho\sigma}$. Since $x\eta = [\![x\gamma\rho]\!]_{\xi,\sigma}$, by candidate substitution, $[\![\vec{a}]\!]_{\eta,\gamma\rho\sigma} = [\![\vec{a}\gamma\rho]\!]_{\xi,\sigma}$. So, $[\![\vec{a}]\!]_{\eta,\gamma\sigma} = [\![\vec{a}\delta]\!]_{\xi,\sigma} = [\![\vec{a}\delta]\!]_{\xi\xi',\sigma\sigma'} = [\![\vec{a}]\!]_{\xi'',\delta\sigma\sigma'}$. Now, by induction hypothesis, $\vec{u}'\sigma\sigma' \in [\![\vec{U}'\delta']\!]_{\xi\xi',\sigma\sigma'}$. Therefore, since $l_i\sigma = l_i\sigma\sigma' \in [\![T_i\gamma\rho]\!]_{\xi,\sigma} = [\![T_i\gamma\rho]\!]_{\xi\xi',\sigma\sigma'}$, by Lemma 54, $u_j\sigma\sigma' \in [\![U_j\delta\rho]\!]_{\xi\xi',\sigma\sigma'}$ and $o_{C(\vec{R})}(l_i\sigma) > o_{C(\vec{R})}(u_j\sigma\sigma')$ where $\vec{R} = [\![\vec{v}\gamma'\rho]\!]_{\xi\xi',\sigma\sigma'} = \vec{S}$. □

Lemma 68 (Computability of higher-order symbols) For all $f \in \mathcal{F}_\omega$, $f \in [\![\tau_f]\!]$.

Proof. Assume that $f : (\vec{x} : \vec{T})U$. $f \in [\![\tau_f]\!]$ iff, for all Γ_f-valid pair (η,θ), $f\vec{x}\theta \in [\![U]\!]_{\eta,\theta}$. We prove it by induction on $((f,\eta,\theta),\theta)$ with $(\sqsupset,\to)_{\mathrm{lex}}$ as well-founded ordering. Let $t_i = x_i\theta$ and $t = f\vec{t}$. By assumption (see Definition 2), for all rule $f\vec{l} \to r \in \mathcal{R}$, $|\vec{l}| \leq |\vec{t}|$. So, if $U \neq C\vec{v}$ with $C \in \mathcal{CF}^\Box$ then t is neutral and it suffices to prove that $\to(t) \subseteq [\![U]\!]_{\eta,\theta}$. Otherwise, $[\![U]\!]_{\eta,\theta} = I_C(\vec{a})$ with $a_i = (v_i\theta, [\![v_i]\!]_{\eta,\theta})$. Since $\eta,\theta \models \Gamma_f$, $t_j \in [\![T_j]\!]_{\eta,\theta}$. Therefore, in this case too, it suffices to prove that $\to(t) \subseteq [\![U]\!]_{\eta,\theta}$.

If the reduction takes place in one t_i then we can conclude by induction hypothesis since reducibility candidates are stable by reduction and $\sqsupset$ is compatible with reduction. Assume now that there exist $(l \to r,\Gamma,\rho) \in \mathcal{R}$ and σ such that $l = f\vec{l}$ and $t = l\sigma$. Then, $\theta = \gamma\sigma$ with $\gamma = \{\vec{x} \mapsto \vec{l}\}$. Furthermore, by **(S5)**, $\sigma \downarrow \rho\sigma$. Hence, by Lemma 38, $[\![U]\!]_{\eta,\theta} = [\![U]\!]_{\eta,\gamma\rho\sigma}$ and $[\![\vec{T}]\!]_{\eta,\theta} = [\![\vec{T}]\!]_{\eta,\gamma\rho\sigma}$. Now, since rules are well-formed, $\Gamma \vdash l\rho : U\gamma\rho$. Therefore, by inversion, $\Gamma \vdash l_i\rho : T_i\gamma\rho$ and $\gamma\rho : \Gamma_f \rightsquigarrow \Gamma$.

We now define ξ such that $[\![U]\!]_{\eta,\gamma\rho\sigma} = [\![U\gamma\rho]\!]_{\xi,\sigma}$ and $[\![\vec{T}]\!]_{\eta,\gamma\rho\sigma} = [\![\vec{T}\gamma\rho]\!]_{\xi,\sigma}$. By safeness **(b)**, for all $x \in \mathrm{FV}^\Box(\vec{T}U)$, $x\gamma\rho \in \mathrm{dom}(\Gamma)$ and, for all $x,x' \in \mathrm{FV}^\Box(\vec{T}U)$, $x\gamma\rho = x'\gamma\rho \Rightarrow x = x'$. Let $y \in \mathrm{dom}^\Box(\Gamma)$. If there exists $x \in \mathrm{dom}(\Gamma_f)$ (necessarily unique) such that $y = x\gamma\rho$, we take $y\xi = x\eta$. Otherwise, we take $y\xi = \top_{y\Gamma}$. We check that $\xi \models \Gamma$. If $y \neq x\gamma\rho$, $y\xi = \top_{y\Gamma} \in \mathcal{R}_{y\Gamma}$. If $y = x\gamma\rho$ then $y\xi = x\eta$. Since $\eta \models \Gamma_f$, $x\eta \in \mathcal{R}_{x\Gamma_f}$. Since $\gamma\rho : \Gamma_f \rightsquigarrow \Gamma$, $\Gamma \vdash y : x\Gamma_f\gamma\rho$. Therefore, $y\Gamma\ \mathbb{C}_\Gamma^*\ x\Gamma_f\gamma\rho$ and, by Lemma 34, $y\xi = x\eta \in \mathcal{R}_{x\Gamma_f} = \mathcal{R}_{x\Gamma_f\gamma\rho} = \mathcal{R}_{y\Gamma}$. So, $\xi \models \Gamma$. Now, by candidate substitution, $[\![U\gamma\rho]\!]_{\xi,\sigma} = [\![U]\!]_{\xi',\gamma\rho\sigma}$ with $x\xi' = [\![x\gamma\rho]\!]_{\xi,\sigma}$. Let $x \in \mathrm{FV}(\vec{T}U)$. By **(b)**, $x\gamma\rho = y \in \mathrm{dom}^\Box(\Gamma)$ and $x\xi' = y\xi = x\eta$. Since ξ' and η are equal on $\mathrm{FV}^\Box(\vec{T}U)$, $[\![U]\!]_{\xi',\gamma\rho\sigma} = [\![U]\!]_{\eta,\gamma\rho\sigma} = [\![U\gamma\rho]\!]_{\xi,\sigma}$ and $[\![\vec{T}]\!]_{\xi',\gamma\rho\sigma} = [\![\vec{T}]\!]_{\eta,\gamma\rho\sigma} = [\![\vec{T}\gamma\rho]\!]_{\xi,\sigma}$.

We now prove that σ is adapted to ξ. Let $x \in \mathrm{dom}(\Gamma)$. Since rules are well-formed, there exists i such that $l_i : T_i\gamma \rhd_\rho^* x : x\Gamma$ and $\mathrm{dom}(\rho) \subseteq \mathrm{FV}(l) \setminus \mathrm{dom}(\Gamma)$. Since $l_i\sigma \in [\![T_i\gamma\rho]\!]_{\xi,\sigma}$, by correctness of accessibility, $x\sigma \in [\![x\Gamma\rho]\!]_{\xi,\sigma}$. Since $\mathrm{dom}(\rho) \cap \mathrm{dom}(\Gamma) = \emptyset$, $x\Gamma\rho = x\Gamma$

and $x\sigma \in [\![x\Gamma]\!]_{\xi,\sigma}$. Therefore, σ is adapted to ξ and, by correctness of the computability closure, $r\sigma \in [\![U\gamma\rho]\!]_{\xi,\sigma} = [\![U]\!]_{\eta,\theta}$. □

Lemma 69 (Computability of well-typed terms) If $\Gamma \vdash t : T$ and $\xi, \theta \models \Gamma$ then $t\theta \in [\![T]\!]_{\xi,\theta}$.

Proof. After Lemmas 63, 66 and 68. □

Theorem 70 (Strong normalization) Every typable term is strongly normalizable.

Proof. Assume that $\Gamma \vdash t : T$. Let $x\xi = \top_{x\Gamma}$ for all $x \in \mathrm{dom}(\Gamma)$. Since $\xi \models \Gamma$ and the identity substitution ι is adapted to ξ, $t \in S = [\![T]\!]_{\xi,\iota}$. Now, either $T = \Box$ or $\Gamma \vdash T : s$ for some s. If $T = \Box$ then $S = \top_\Box = \mathcal{SN}$. If $\Gamma \vdash T : s$ then $S \in \mathcal{R}_s$ and $S \subseteq \mathcal{SN}$ by **(R1)**. So, in both cases, $t \in \mathcal{SN}$. □

7. Future directions of research

We conclude by giving some directions of research for improving our conditions of strong normalization.

Rewriting modulo. We did not consider rewriting modulo some equational theories like associativity and commutativity. While this does not create too much difficulties at the object level (Blanqui 2003, RTA), it is less clear for rewriting at the type level.

Quotient types. We have seen that rewrite rules on constructors allows us to formalize some quotient types. However, to prove properties by induction on such types requires to know what the normal forms are (Jouannaud and Kounalis 1986) and may also require a particular reduction strategy (Courtieu 2001) or conditional rewriting.

Confluence. Among our strong normalization conditions, we not only require rewriting to be confluent but also its combination with β-reduction. This is a strong condition since we cannot rely on strong normalization for proving confluence (Nipkow 1991; Blanqui 2000). Except for first-order rewriting systems without dependent types (Breazu-Tannen and Gallier 1994) or left-linear higher-order rewrite systems (Müller 1992; Van Oostrom 1994), few results are known on modularity of confluence for the combination of higher-order rewriting and β-reduction. Therefore, it would be interesting to study this problem more deeply.

Local confluence. We believe that local confluence is sufficient for establishing strong normalization since local confluence and strong normalization together imply confluence. But, then, it seems necessary to prove many properties simultaneously (subject reduction, strong normalization and confluence), which seems difficult.

Simplicity. For non-primitive predicate symbols, we require that their defining rules have no critical pairs between them or with the other rules. These strong conditions

allow us to define a valid interpretation in a simple way. It is important to be able to weaken these conditions in order to capture more decision procedures.

Local definitions. In our work, we considered only globally defined symbols, that is, symbols whose type is typable in the empty environment. However, in practice, during a formal proof in a system like Coq (Coq Development Team 2002), it may be very useful to introduce symbols and rules using some hypothesis. We should study the problems arising from local definitions and how our results can be used to solve them. Local abbreviations are studied by Poll and Severi (Poll and Severi 1994) and local definitions by rewriting are considered by Chrzaszcz (Chrząszcz 2000).

HORPO. For higher-order definitions, we have chosen to extend the General Schema of Jouannaud and Okada (Jouannaud and Okada 1997). But the Higher-Order Recursive Path Ordering (HORPO) of Jouannaud and Rubio (Jouannaud and Rubio 1999), which is an extension of RPO to the simply typed λ-calculus, is naturally more powerful. Walukiewicz recently extended this ordering to the Calculus of Constructions with symbols at the object level only (Walukiewicz 2000; Walukiewicz-Chrząszcz 2002). The combination of the two works should allow us to extend RPO to the Calculus of Constructions with type-level rewriting too.

η-Reduction. Among our conditions, we require the confluence of $\rightarrow_{\mathcal{R}} \cup \rightarrow_\beta$. Hence, our results cannot be directly extended to η-reduction, which is well known to create important difficulties (Geuvers 1993) since $\rightarrow_\beta \cup \rightarrow_\eta$ is not confluent on not well-typed terms.

Non-strictly positive predicates. The ordering used in the General Schema for comparing the arguments of function symbols can capture recursive definitions on basic and strictly-positive types, but cannot capture recursive definitions on non-strictly positive types (Matthes 2000). However, Mendler (Mendler 1987) showed that such definitions are strongly normalizing. In (Blanqui 2003, TLCA), we recently showed how to deal with such definitions in the Calculus of Algebraic Constructions.

Acknowledgments: I would like to thank very much Daria Walukiewicz who pointed to me several errors or imprecisions in previous versions of this work. I also thank Jean-Pierre Jouannaud, Gilles Dowek, Christine Paulin, Herman Geuvers, Thierry Coquand and the anonymous referees for their useful comments on previous versions of this work.

References

M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

A. Abel. A third-order representation of the $\lambda\mu$-calculus. In *Proc. of the Workshop on Mechanized Reasoning about Languages with Variable Binding*, Electronic Notes in Theoretical Computer Science, 58(1), 2001.

T. Altenkirch. *Constructions, Inductive Types and Strong Normalization*. PhD thesis, University of Edinburgh, United Kingdom, 1993.

L. Augustsson. Compiling pattern matching. In *Proc. of the Conf. on Functional Programming Languages and Computer Architecture*, LNCS 201, 1985.

F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

F. Barbanera. Adding algebraic rewriting to the Calculus of Constructions: strong normalization preserved. In *Proc. of the 2nd Int. Work. on Conditional and Typed Rewriting Systems*, LNCS 516, 1990.

F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization and confluence in the algebraic-λ-cube. In *Proc. of the 9th IEEE Symp. on Logic in Computer Science*, 1994.

F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization in the algebraic-λ-cube. *Journal of Functional Programming*, 7(6):613–660, 1997.

H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 2nd edition, 1984.

H. Barendregt. Lambda calculi with types. In S. Abramski, D. Gabbay, and T. Maibaum, editors, *Handbook of logic in computer science*, volume 2. Oxford University Press, 1992.

B. Barras. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, Université Paris VII, France, 1999.

G. Barthe. The relevance of proof-irrelevance. In *Proc. of the 25th Int. Colloq. on Automata, Languages and Programming*, LNCS 1443, 1998.

G. Barthe and H. Geuvers. Modular properties of algebraic type systems. In *Proc. of the 2nd Int. Work. on Higher-Order Algebra, Logic and Term Rewriting*, LNCS 1074, 1995.

G. Barthe and H. Geuvers. Congruence types. In *Proc. of the 9th Int. Work. on Computer Science Logic*, LNCS 1092, 1995.

G. Barthe and P.-A. Melliès. On the subject reduction property for algebraic type systems. In *Proc. of the 10th Int. Work. on Computer Science Logic*, LNCS 1258, 1996.

G. Barthe and F. van Raamsdonk. Termination of algebraic type systems: the syntactic approach. In *Proc. of the 6th Int. Conf. on Algebraic and Logic Programming*, LNCS 1298, 1997.

P. Bendix and D. Knuth. *Computational problems in abstract algebra*, chapter Simple word problems in universal algebra. Pergamon Press, 1970.

F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Proc. of the 11th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1833, 2000.

F. Blanqui. *Théorie des Types et Récriture*. PhD thesis, Université Paris XI, Orsay, France, 2001. Available in english as "Type Theory and Rewriting".

F. Blanqui. Definitions by rewriting in the Calculus of Constructions (extended abstract). In *Proc. of the 16th IEEE Symp. on Logic in Computer Science*, 2001.

F. Blanqui. Inductive types in the Calculus of Algebraic Constructions. In *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 2701, 2003.

F. Blanqui. Rewriting modulo in Deduction modulo. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003.

F. Blanqui, J.-P. Jouannaud, and M. Okada. The Calculus of Algebraic Constructions. In *Proc. of the 10th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1631, 1999.

F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type Systems. *Theoretical Computer Science*, 272:41–68, 2002.

P. Borovanský, H. Cirstea, H. Dubois, C. Kirchner, H. Kirchner, P.-E. Moreau, C. Ringeissen, and M. Vittek. *ELAN User Manual*. INRIA Nancy, France, 2000. `http://elan.loria.fr/`.

V. Breazu-Tannen. Combining algebra and higher-order types. In *Proc. of the 3rd IEEE Symp. on Logic in Computer Science*, 1988.

V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. In *Proc. of the 16th Int. Colloq. on Automata, Languages and Programming*, LNCS 372, 1989.

V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic confluence. *Information and Computation*, 114(1):1–29, 1994.

J. Chrząszcz. Modular rewriting in the Calculus of Constructions, 2000. Presented at the Int. Work. on Types for Proofs and Programs.

M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and Programming in Rewriting Logic*. Computer Science Laboratory, SRI International, United States, 1999. `http://maude.csl.sri.com/`.

E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME, 2000. `http://cime.lri.fr/`.

Coq Development Team. *The Coq Proof Assistant Reference Manual – Version 7.3*. INRIA Rocquencourt, France, 2002. `http://coq.inria.fr/`.

T. Coquand. *Une théorie des constructions*. PhD thesis, Université Paris VII, France, 1985.

T. Coquand. An analysis of Girard's paradox. In *Proc. of the 1st IEEE Symp. on Logic in Computer Science*, 1986.

T. Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

T. Coquand. Pattern matching with dependent types. In *Proc. of the Int. Work. on Types for Proofs and Programs*, , 1992. `http://www.lfcs.informatics.ed.ac.uk/research/types-bra/proc/`.

T. Coquand and J. Gallier. A proof of strong normalization for the Theory of Constructions using a Kripke-like interpretation, 1990. Paper presented at the 1st Int. Work. on Logical Frameworks but not published in the proceedings. Available at `ftp://ftp.cis.upenn.edu/pub/papers/gallier/sntoc.dvi.Z`.

T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2–3):95–120, 1988.

T. Coquand and C. Paulin-Mohring. Inductively defined types. In *Proc. of the Int. Conf. on Computer Logic*, LNCS 417, 1988.

P. Courtieu. Normalized types. In *Proc. of the 15th Int. Work. on Computer Science Logic*, LNCS 2142, 2001.

N. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In *Proc. of the Symp. on Automatic Demonstration*, Lecture Notes in Mathematics. Springer, 1968. Reprinted in (Geuvers *et al.* 1994).

N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

N. Dershowitz. Hierarchical termination. In *Proc. of the 4th Int. Work. on Conditional and Typed Rewriting Systems*, LNCS 968, 1994.

N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6. North-Holland, 1990.

D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. In *Proc. of the 4th Int. Conf. on Rewriting Techniques and Applications*, LNCS 488, 1991.

G. Dowek. La part du calcul, 1999. Mémoire d'habilitation.

G. Dowek and B. Werner. Proof normalization modulo. In *Proc. of the Int. Work. on Types for Proofs and Programs, , LNCS 1657, 1998.*

G. Dowek and B. Werner. An inconsistent theory modulo defined by a confluent and terminating rewrite system, 2000.

G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. Technical Report 3400, INRIA Rocquencourt, France, 1998.

G. Dowek, T. Hardin, and C. Kirchner. HOL-lambda-sigma: an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11:1–25, 2001.

K. Drosten. *Termersetzungssysteme.* PhD thesis, Universität Passau, Germany, 1989.

S. Eker. Fast matching in combinations of regular equational theories. In *Proceedings of the 1st International Workshop on Rewriting Logic and Applications*, Electronic Notes in Theoretical Computer Science 4, 1996.

M. Fernández. *Modèles de calculs multiparadigmes fondés sur la réécriture.* PhD thesis, Université Paris XI, Orsay, France, 1993.

J. Gallier. On Girard's "Candidats de Réductibilité". In P.-G. Odifreddi, editor, *Logic and Computer Science.* North-Holland, 1990.

H. Geuvers. A short and flexible proof of strong normalization for the Calculus of Constructions. In *Proc. of the Int. Work. on Types for Proofs and Programs, , LNCS 996, 1994.*

H. Geuvers. *Logics and Type Systems.* PhD thesis, Katholiecke Universiteit Nijmegen, The Netherlands, 1993.

H. Geuvers and M.-J. Nederhof. A modular proof of strong normalization for the Calculus of Constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.

H. Geuvers, R. Nederpelt, and R. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1994.

E. Giménez. *Un Calcul de Constructions infinies et son application à la vérification de systèmes communiquants.* PhD thesis, ENS Lyon, France, 1996.

E. Giménez. Structural recursive definitions in type theory. In *Proc. of the 25th Int. Colloq. on Automata, Languages and Programming*, LNCS 1443, 1998.

J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. Fenstad, editor, *Proc. of the 2nd Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1971.

J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmetique d'ordre supérieur.* PhD thesis, Université Paris VII, France, 1972.

J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1988.

B. Grégoire and X. Leroy. A compiled implementation of strong reduction. In *Proceedings of the 7th ACM SIGPLAN International Conference on Functional Programming*, 2002.

J. Guttag and J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10(1):27–52, 1978.

R. Harper and J. Mitchell. Parametricity and variants of Girard's J operator. *Information Processing Letters*, 70:1–5, 1999.

J. Hsiang. *Topics in Automated Theorem Proving and Program Generation.* PhD thesis, University of Illinois, Urbana-Champaign, United States, 1982.

J.-P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proc. of the 1st IEEE Symp. on Logic in Computer Science*, 1986.

J.-P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proc. of the 6th IEEE Symp. on Logic in Computer Science*, 1991.

J.-P. Jouannaud and M. Okada. Abstract Data Type Systems. *Theoretical Computer Science*, 173(2):349–391, 1997.

J.-P. Jouannaud and A. Rubio. The Higher-Order Recursive Path Ordering. In *Proc. of the 14th IEEE Symp. on Logic in Computer Science*, 1999.

H. Kirchner and P.-E. Moreau. Promoting rewriting to a programming language: A compiler for non-deterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, 11(2):207–251, 2001.

J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

E. Kounalis. Completeness in data type specifications. In *Proc. of the European Conf. on Computer Algebra*, LNCS 204, 1985.

Z. Luo and R. Pollack. *LEGO Proof Development System: User's manual*. University of Edinburgh, United Kingdom, 1992. `http://www.dcs.ed.ac.uk/home/lego/`.

P. Martin-Löf. *Intuitionistic type theory*. Bibliopolis, Napoli, Italy, 1984.

R. Matthes. Lambda calculus: A case for inductive definitions, 2000.

N. P. Mendler. *Inductive Definition in Type Theory*. PhD thesis, Cornell University, United States, 1987.

F. Müller. Confluence of the lambda calculus with left-linear algebraic rewriting. *Information Processing Letters*, 41(6):293–299, 1992.

R. Nederpelt. *Strong normalization in a typed lambda calculus with lambda structured types*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1973.

T. Nipkow. Higher-order critical pairs. In *Proc. of the 6th IEEE Symp. on Logic in Computer Science*, 1991.

M. Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. In *Proc. of the 1989 Int. Symp. on Symbolic and Algebraic Computation, , ACM Press*.

C. Paulin-Mohring. Extracting Fω's programs from proofs in the Calculus of Constructions. In *Proc. of the 16th ACM Symp. on Principles of Programming Languages, 1989*.

G. Peterson and M. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.

D. A. Plaisted. A recursively defined ordering for proving termination of term rewriting systems. Technical report, University of Illinois, Urbana-Champaign, United States, 1978.

E. Poll and P. Severi. Pure Types Systems with definitions. In *Proc. of the 3rd Int. Symp. on Logical Foundations of Computer Science*, LNCS 813, 1994.

J. Reynolds. Types, abstraction and parametric poymorphism. In *Proc. of the 9th IFIP World Computer Congress*, North-Holland, 1983.

M. Rusinowitch. On termination of the direct sum of term-rewriting systems. *Information Processing Letters*, 1987.

M. P. A. Sellink. Verifying process algebra proofs in type theory. In *Proc. of the Int. Work. on Semantics of Specification Languages*, Workshops in Computing, 1993.

M. Stefanova. *Properties of Typing Systems*. PhD thesis, Katholiecke Universiteit Nijmegen, The Netherlands, 1998.

W. W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

J.-J. Thiel. Stop loosing sleep over incomplete specifications. In *Proc. of the 11th ACM Symp. on Principles of Programming Languages, 1984*.

Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.

J. van de Pol. Termination proofs for higher-order rewrite systems. In *Proc. of the 1st Int. Work. on Higher-Order Algebra, Logic and Term Rewriting*, LNCS 816, 1993.

J. van de Pol. *Termination of higher-order rewrite systems.* PhD thesis, Utrecht Universiteit, Nederlands, 1996.

J. van de Pol and H. Schwichtenberg. Strict functionals for termination proofs. In *Proc. of the 2nd Int. Conf. on Typed Lambda Calculi and Applications*, LNCS 902, 1995.

V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit Amsterdam, The Netherlands, 1994.

D. Walukiewicz. Termination of rewriting in the Calculus of Constructions (extended abstract). In *Proc. of the Workshop on Logical Frameworks and Meta-languages*, 2000.

D. Walukiewicz-Chrząszcz. Termination of rewriting in the Calculus of Constructions. *Journal of Functional Programming*, 2002. To appear.

B. Werner. *Une Théorie des Constructions Inductives.* PhD thesis, Université Paris VII, France, 1994.

H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, 1994.